HealthBot+
Software Architecture Document

Version 1.0

210329E - LAKSARA K.Y.
210333K - LAKSHAN P.D.
210343P - LIYANAGE I.V.S.

Revision History

Date	Version	Description	Author
17/7/2024	1.0	Initial Commit	210329E, 210333K, 210343P

Table of Contents

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the HealthBot+ system, utilizing various architectural views to depict different aspects of the system. It aims to capture and convey the significant architectural decisions made for the system. This document is intended for software architects, developers, and stakeholders to understand the architectural framework and design considerations of HealthBot+.

1.2 Scope

The Software Architecture Document applies to the HealthBot+ AI-powered web application, which focuses on the early detection and management of skin diseases. The document outlines the architecture that influences the development, deployment, and maintenance of the system, ensuring scalability, reliability, and security.

1.3 Definitions, Acronyms, and Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DICOM	Digital Imaging and Communications in Medicine
XAI	Explainable Artificial Intelligence
AWS	Amazon Web Services
React	A JavaScript library for building user interfaces
Django	A high-level Python web framework
SIIM-ISIC	Society for Imaging Informatics in Medicine - International Skin Imaging Collaboration

1.4 References

1) SkinVision. (2024, May 30). SkinVision | Skin Cancer Melanoma Detection App | SkinVision. SkinVision - https://www.skinvision.com

2) U. K. Lilhore, S. Simaiya, Y. K. Sharma, K. S. Kaswan, K. B. V. B. Rao, V. V. R. M. Rao, A. Baliyan, A. Bijalwan, and R. Alroobaea, "A precise model for skin cancer diagnosis using hybrid U-Net and improved MobileNet-V3 with hyperparameters optimization," Scientific Reports, vol. 14, no. 4299, 2024. doi: 10.1038/s41598-024-54212-8.

3) J. J. Bolognia, J. L. Jorizzo, and J. V. Schaffer, Dermatology, 4th ed. Philadelphia, PA: Elsevier Saunders, 2017.

Tools used for diagrams :

***LucidChart** - https://www.lucidchart.com*

2. Architectural Representation

This section describes the software architecture for the HealthBot+ system and how it is represented through different views.

The software architecture of HealthBot+ is designed to facilitate the accurate detection and management of skin diseases through an AI-powered web application. It leverages cloud computing services and integrates advanced machine learning models to provide diagnostic support.

2.1 Use-Case View

The Use-Case view identifies the primary interactions between users (patients, healthcare professionals mainly doctors) and the system. Key use cases include:

- **Image Upload and Analysis**: Users upload skin lesion images for automated analysis.
- **Chatbot Interaction**: Users interact with an oncology-trained chatbot for medical advice and support.
- **Report Generation**: Automated generation of medical reports based on diagnostic results.

2.2 Logical View

The Logical view defines the system's functionality in terms of modules and components:

- **Presentation Layer**: User interface components for interacting with the system.
- **Application Layer**: Contains business logic for image analysis, chatbot interactions, and report generation.

- **Data Layer**: Manages storage and retrieval of image data, patient information, and diagnostic results.

2.3 Process View

The Process view illustrates how concurrent processes and tasks are executed within the system:

- **Image Processing Pipeline**: Sequence of tasks for image upload, preprocessing, feature extraction, and classification.
- **Chatbot Interaction Workflow**: Processes for user input handling, query processing, and response generation.
- **Report Generation Process**: Steps involved in generating medical reports based on analysis outcomes.

2.4 Deployment View

The Deployment view specifies how the system is distributed across computing nodes and resources:

- **Cloud Deployment**: Utilizes Azure or AWS for scalable and reliable cloud infrastructure.
- **Service Components**: Deployment of AI models, databases, and web application services.
- **Load Balancing and Scalability**: Ensures performance and availability through load balancing and scalable deployment configurations.

2.5 Implementation View

The Implementation view focuses on the physical components and technologies used:

- **Programming Languages**: Python for AI model development, JavaScript for frontend interactions.
- **Frameworks and Libraries**: TensorFlow/Keras for deep learning, Flask for backend API development, React for frontend UI.
- **Integration with AI Services**: Utilizes pretrained models for skin disease classification and natural language processing (NLP) for chatbot functionalities.

3. Architectural Goals and Constraints

This section outlines the software requirements and objectives that significantly impact the architecture of HealthBot+. It also identifies special constraints that affect design and implementation.

3.1 Software Requirements and Objectives

3.1.1 Safety

- **Accuracy**: Ensuring high accuracy in diagnosing skin diseases to avoid misdiagnosis.
- **Error Handling**: Implementing robust error handling mechanisms to manage incorrect or incomplete data inputs.

3.1.2 Security

- **Data Encryption**: Encrypting sensitive patient information both in transit and at rest to protect against unauthorized access.
- **Authentication**: Implementing strong user authentication and authorization mechanisms to ensure that only authorized personnel can access specific functionalities.

3.1.3 Privacy

- **Compliance**: Ensuring compliance with privacy regulations such as HIPAA and GDPR to protect patient data.
- **Anonymization**: Anonymizing patient data to ensure privacy during data analysis and storage.

3.1.4 Portability

- **Platform Independence**: Designing the system to be platform-independent, allowing it to run on various operating systems and devices.
- **Modular Architecture**: Using a modular architecture to facilitate portability and ease of deployment across different environments.

3.1.5 Distribution

- **Scalability**: Ensuring the system can scale to handle a large number of concurrent users without performance degradation.
- **Cloud Integration**: Leveraging cloud services (e.g., AWS or Azure) for distributed processing and storage.

3.1.6 Reuse

- **Pretrained Models**: Utilizing pretrained models and fine-tuning them for specific use cases to accelerate development and improve accuracy.
- **Code Reusability**: Adopting design patterns and coding practices that promote code reusability across different modules.

3.2 Special Constraints

3.2.1 Design and Implementation Strategy

- **Agile Methodology**: Following an Agile development methodology to allow iterative development, regular feedback, and continuous improvement.
- **Microservices Architecture**: Employing a microservices architecture to enhance scalability, maintainability, and ease of deployment.

3.2.2 Development Tools

- **Integrated Development Environment (IDE)**: Using industry-standard IDEs such as Visual Studio Code and Google Colab for efficient development.
- **Version Control**: Implementing Git for version control to track changes and facilitate collaborative development.
- **Kaggle Notebooks**: Utilizing Kaggle Notebooks for prototyping and collaborative machine learning model development.

3.2.3 Team Structure

- **Unified Team:** All three members collaboratively contribute to different aspects of development, including AI, machine learning, frontend and backend.
- **Collaboration Tools:** Utilizing GitHub for version control and collaboration to manage code, track changes, and facilitate teamwork.

3.2.4 Schedule

- **Project Phases**: Dividing the project into well-defined phases with specific milestones to track progress and ensure timely delivery.
- **Time Management**: Allocating sufficient time for each phase, including development, testing, and deployment, to avoid rushed or incomplete work.

3.2.5 Legacy Code

- **Integration:** Ensuring seamless integration with existing systems, including the integration of ChatGPT-3.5 Turbo, to avoid disruptions and maintain continuity.
- **Refactoring**: Refactoring any legacy code where necessary to align with current design principles and coding standards, ensuring compatibility with modern systems and technologies.

4. Use-Case View

The Use-Case View outlines the primary functionalities of HealthBot+ that are pivotal to its architecture and operational scope.

Here are the identified Use Cases for HealthBot+;

Common Use Cases for All Actors

- Registration
- Login to the System
- Logout from the System
- Edit Profile Details
- View Personal Details
- Edit Personal Details
- Access FAQs

Use Cases for Doctor

- Edit Patient Reports
- View Patient Details
- View Patient Reports
- Create Patient Prescriptions
- Edit Patient Prescriptions

Use Cases for Patient

- Image Upload
- Chatbot Interaction
- View Patient Reports
- View Prescriptions
- View Consultation Details

Use Cases for Administrator

- System Monitoring
- Check Feasibility of Doctors
- Manage User Accounts

4.1 Use-Case Realizations

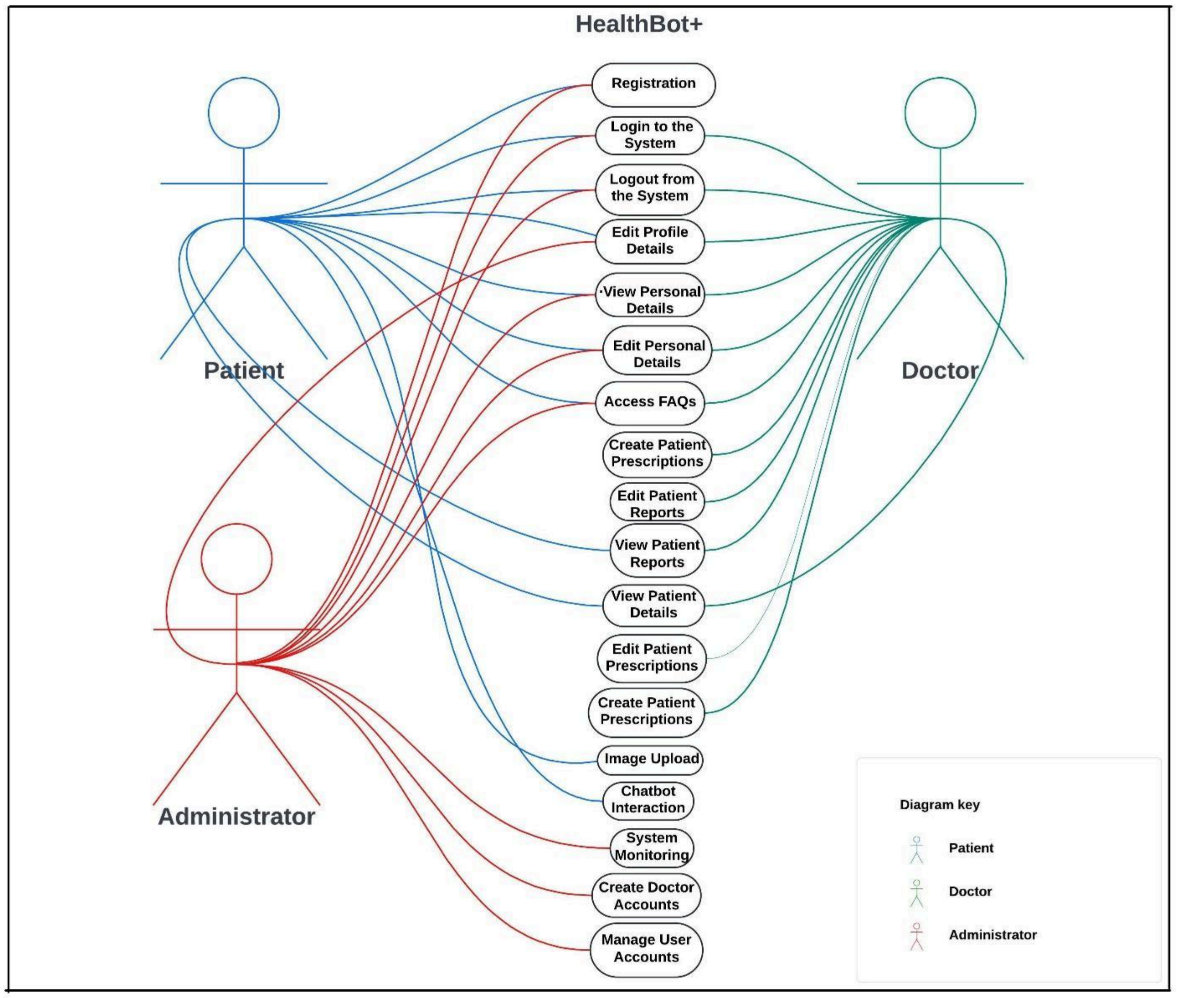


Fig:1 Use-Case Diagram

4.1.1

Use case name	Registration
Actor	Patient, Administrator
Description	Allows actors to create a new account within the HealthBot+ system.
preconditions	None
Main flow	•Actor navigates to the registration page.

	•Actor fills out a registration form with required details (e.g., name, email, password). •System validates entered information. •System creates a new account for the actor.
Successful end/post condition	Actor's account is successfully registered in the system.
Fail end/post condition	Registration fails due to existing email or invalid information.
Extensions	N/A

4.1.2

Use case name	Login to the System
Actor	Doctor, Patient, Administrator
Description	Enables actors to securely access the HealthBot+ system.
preconditions	Actors must have valid credentials (username and password).
Main flow	•Actor navigates to the login page. •Actor enters the username and password. •System verifies credentials. •System authenticates the actor and grants access to the system.
Successful end/post condition	Actors successfully logged into the system and can proceed to use case-specific functionalities.
Fail end/post condition	Invalid credentials result in access denial.
Extensions	**Update Password:** Actor can update a password

4.1.3

Use case name	Logout from the System
Actor	Doctor, Patient, Administrator
Description	Allows actors to securely log out from the HealthBot+ system.

preconditions	Actors must be logged into the system.
Main flow	•Actor navigates to the logout option in the system. •System logs out the actor, clearing session data.
Successful end/post condition	Actor is successfully logged out from the system.
Fail end/post condition	Logout fails due to technical issues.
Extensions	N/A

4.1.4

Use case name	Edit Profile Details
Actor	Doctor, Patient, Administrator
Description	Enables actors to modify their personal information stored in the system.
preconditions	Actors must be logged into the system.
Main flow	•Actor navigates to the profile settings section. •Actor selects the option to edit profile details. •Actors update relevant information (e.g., contact details, preferences). •System validates and saves the updated information.
Successful end/post condition	Actor's profile details are successfully updated.
Fail end/post condition	Update fails due to validation errors.
Extensions	**Profile Picture Update:** Actors can upload or change profile pictures.

4.1.5

Use case name	View Personal Details
Actor	Doctor, Patient, Administrator
Description	Allows actors to review their personal information stored in the system..
preconditions	Actors must be logged into the system.

Main flow	•Actor navigates to the personal details section. •System retrieves and displays the actor's personal information (e.g., name, sex ,age,..)
Successful end/post condition	Actors can view their personal details.
Fail end/post condition	Personal details retrieval fails due to system error.
Extensions	N/A

4.1.6

Use case name	Access FAQs
Actor	Doctor, Patient, Administrator
Description	Allows users to view frequently asked questions (FAQs) to find quick answers to common queries about the system.
preconditions	User is logged into the system.
Main flow	•User navigates to the FAQ section from the main menu. •System displays a list of frequently asked questions organized by categories. •User selects a question to view the detailed answer. •System displays the answer to the selected question.
Successful end/post condition	User finds and views the answer to their question from the FAQ section.
Fail end/post condition	User cannot find the answer to their question due to it not being listed or technical issues.
Extensions	•**Search FAQs:** User can use a search bar to find specific questions and answers quickly. •**Submit Question:** If the user's question is not listed, they can submit it for consideration to be added to the FAQ section.

4.1.7

Use case name	View Prescriptions
Actor	Patient, Doctor
Description	Allows patients to view their prescriptions.
preconditions	Patient is logged into the system.
Main flow	•Patient accesses the prescriptions page. •System displays the prescriptions..
Successful end/post condition	Prescriptions are displayed.
Fail end/post condition	prescription retrieval fails due to data inconsistency.
Extensions	N/A

4.1.9

Use case name	Edit Patient Reports
Actor	Doctor
Description	Allows doctors to edit and update patient medical reports.
preconditions	•Doctor is logged into the system. •Initial patient report must have been generated by the system.
Main flow	•Doctor accesses the patient's medical records. •Doctor selects the report to be edited. •Doctor makes necessary updates and saves changes.
Successful end/post condition	Patient's medical report is successfully updated.
Fail end/post condition	Editing fails due to permission issues.
Extensions	**Addendum to Report:** Doctor can add additional notes or corrections to the report.

4.1.10

Use case name	View Patient Reports
Actor	Doctor, Patient

Description	Allows doctors and patients to access and review medical reports.
preconditions	Actor is logged into the system.
Main flow	•Actor navigates to the patient reports section. •Actor selects a specific report to view. •System retrieves and displays the selected report.
Successful end/post condition	Actor successfully views the patient's medical report.
Fail end/post condition	Report viewing fails due to access restrictions.
Extensions	N/A

4.1.11

Use case name	View Patient Details
Actor	Doctor
Description	Allows doctors to view detailed information about registered patients.
preconditions	Doctor is logged into the system.
Main flow	•Doctor navigates to the patient details section. •Doctor searches for and selects a specific patient. •System retrieves and displays the patient's demographic and medical information.
Successful end/post condition	Doctors can access patient's details for medical assessment.
Fail end/post condition	Patient details retrieval fails due to data inconsistency.
Extensions	N/A

4.1.12

Use case name	Create Patient Prescription
Actor	Doctor

Description	Enables doctors to create medical prescriptions for patients.
preconditions	Doctor is logged into the system.
Main flow	•Doctor selects the patient requiring a prescription. •Doctor enters prescription details (e.g., medication, dosage, instructions). •System validates prescription details and saves the prescription.
Successful end/post condition	Patient's prescription is successfully created and saved.
Fail end/post condition	Prescription creation fails due to medication conflicts or input errors.
Extensions	**Modify Prescription:** Doctor can modify prescription details if needed.

4.1.14

Use case name	Edit Patient Prescription
Actor	Doctor
Description	Allows doctors to modify existing patient prescriptions.
preconditions	Doctor is logged into the system.
Main flow	•Doctor accesses the patient's prescription records. •Doctor selects the prescription to be edited. •Doctor makes necessary changes and saves the updated prescription.
Successful end/post condition	Patient's prescription is successfully updated.
Fail end/post condition	Editing fails due to prescription status or system error.
Extensions	**Revoke Prescription:** Doctor can revoke or cancel the prescription if required.

4.1.15

Use case name	Image Upload
Actor	Patient

Description	Allows patients to upload skin lesion images for diagnostic analysis.
preconditions	Patient is logged into the system.
Main flow	•Patient navigates to the image upload section. •Patient selects and uploads skin lesion images. •System processes and analyzes the uploaded images.
Successful end/post condition	Images are successfully uploaded and queued for analysis.
Fail end/post condition	Images are successfully uploaded and queued for analysis.
Extensions	**Image Analysis Results:** Patient receives a report based on image analysis.

4.1.16

Use case name	Chatbot Interaction
Actor	Patient
Description	Allows patients to interact with an AI-powered chatbot for medical queries and assistance.
preconditions	Patient is logged into the system.
Main flow	•Patient accesses the chatbot interface. •Patient enters a medical query or symptom details. •Chatbot processes the query and provides relevant responses or recommendations.
Successful end/post condition	Patient receives satisfactory assistance or information from the chatbot.
Fail end/post condition	Chatbot unable to understand or provide relevant responses beyond the medical domain.
Extensions	<<N/A>>

4.1.17

Use case name	System Monitoring
Actor	Administrator

Description	Allows administrators to monitor the health and performance of the HealthBot+ system.
preconditions	Administrator is logged into the system.
Main flow	•Administrator accesses the system monitoring dashboard. •Administrator reviews system performance metrics. •System alerts administrator of any critical issues..
Successful end/post condition	System health and performance are monitored effectively.
Fail end/post condition	Dashboard failure prevents real-time monitoring.
Extensions	**Issue Resolution:** Administrator initiates corrective actions for identified issues.

4.1.18

Use case name	Create Doctor Accounts
Actor	Administrator
Description	Allows administrators to create and manage doctor accounts within the system.
preconditions	Administrator is logged into the system.
Main flow	•Administrator navigates to the account management section. •Administrator selects the option to create a new doctor account. •Administrator enters necessary details for the new doctor account (e.g., name, contact information, specialties). •System validates the information and creates the new doctor account. •Administrator assigns necessary permissions and roles to the new doctor account.
Successful end/post condition	A new doctor account is successfully created and added to the system.
Fail end/post condition	Creation of the doctor account fails due to data validation errors or technical issues.
Extensions	**Edit Doctor Account:** Administrator can modify details or permissions of existing doctor accounts.

4.1.19

Use case name	Manage User Accounts
Actor	Administrator
Description	Allows administrators to create, edit, and deactivate user accounts, including both doctor and patient accounts.
preconditions	Administrator is logged into the system.
Main flow	•Administrator navigates to the user account management section. •Administrator selects the option to view all user accounts. •Administrators can choose to create a new user account, edit an existing account, or delete a user account. •For creating a new account: ● Administrator enters necessary details (e.g., name, contact information, role). ● System validates the information and creates the new user account. ● Administrator assigns necessary permissions and roles. •For editing an account: ● Administrator selects an existing account. ● Administrator modifies the necessary details and saves the changes. ● System updates the user account with the new details. •For deleting an account: ● Administrator selects an existing account. ● Administrator deletes the account. ● System deletes the user account and restricts access.

Successful end/post condition	User accounts are successfully created, edited, or deactivated as needed.
Fail end/post condition	Management actions fail due to data validation errors or technical issues.
Extensions	•Role **Assignment:** Administrator can assign or modify roles and permissions for user accounts. •**View Activity Logs:** Administrator can view activity logs related to user account management actions.

5. Logical View

5.1 Overview

The logical view of HealthBot+ is structured into several architecturally significant packages, each representing a different layer or aspect of the system. These packages are designed to encapsulate related functionalities and maintain a clear separation of concerns.

The major packages include:

- **User Management Package:** Handles user registration, authentication, and authorization.
- **Image Processing Package:** Manages the upload, storage, and analysis of medical images.
- **Chatbot Package:** Provides the chatbot functionality for user interaction and assistance.
- **Report Management Package:** Manages the generation, storage, and retrieval of diagnostic reports.
- **Database Access Package:** Encapsulates all interactions with the MongoDB database.
- **Frontend Package:** Contains the React components for the user interface.

Each package is further decomposed into classes and utilities that perform specific tasks within their respective domains.

5.2 Architecturally Significant Design Packages

5.2.1 User Management Package

Description: This package handles all aspects of user management, including registration, login, password management, and authentication.

Diagram:

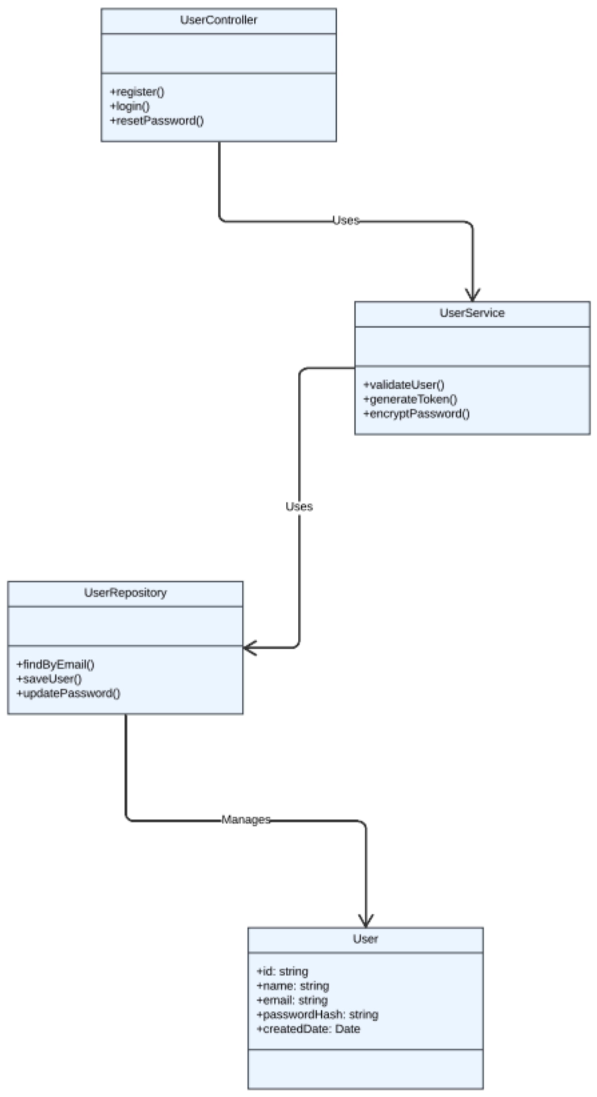


Fig:2 User Management Package Diagram

Classes:

1. UserController

- Description: Manages user-related API endpoints.
- Responsibilities: Handles user registration, login, and password reset requests.
- Operations: register(), login(), resetPassword()

2. UserService

- Description: Provides business logic for user management.
- Responsibilities: Validates user credentials, manages JWT tokens, and handles password encryption.
- Operations: validateUser(), generateToken(), encryptPassword()

3. UserRepository

- Description: Interacts with the MongoDB database to manage user data.
- Responsibilities: Performs CRUD operations on user documents.
- Operations: findByEmail(), saveUser(), updatePassword()

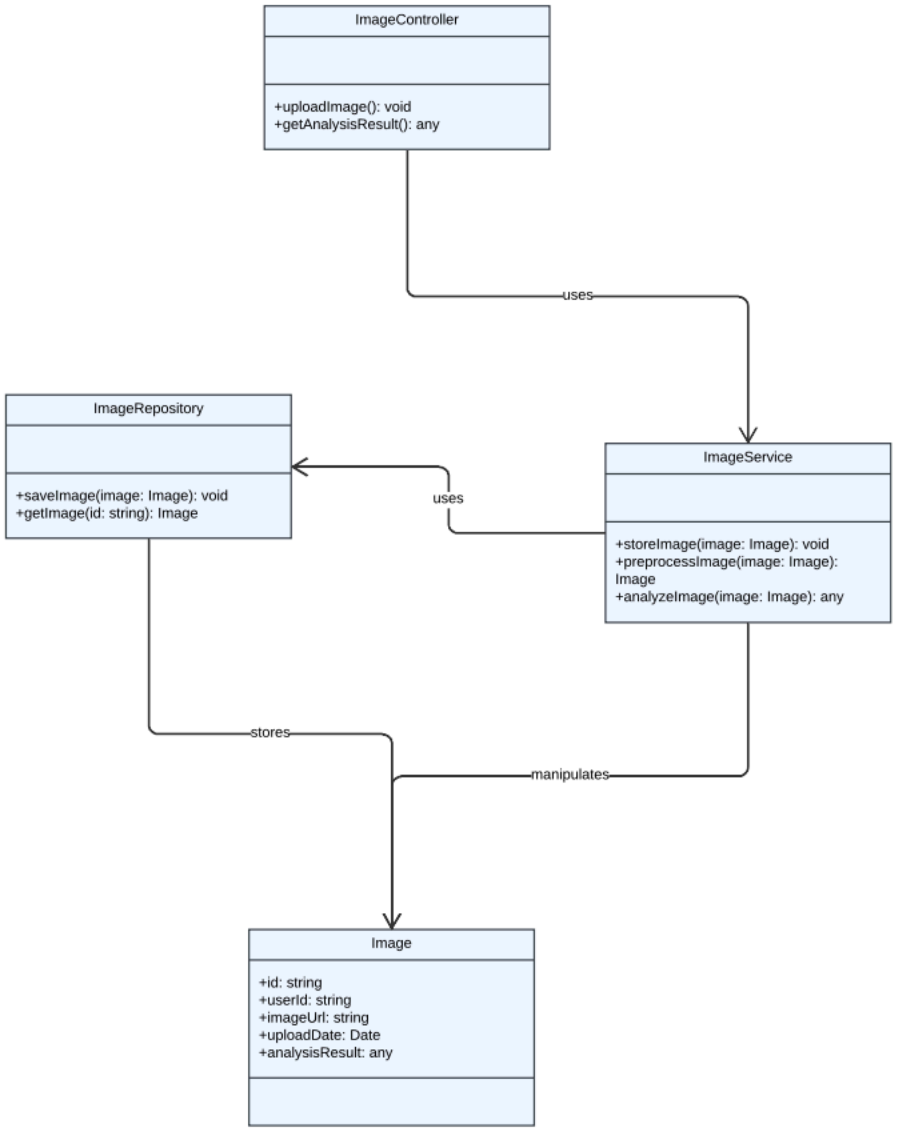
4. User

- Description: Represents the user entity.
- Attributes: id, name, email, passwordHash, createdDate

5.2.2 Image Processing Package

Description: This package handles the upload, storage, and analysis of medical images.

Diagram:



ImageController
+uploadImage(): void
+getAnalysisResult(): any
uses
ImageRepository
+saveImage(image: Image): void
+getImage(id: string): Image
uses
ImageService
+storeImage(image: Image): void
+preprocessImage(image: Image): Image
+analyzeImage(image: Image): any
stores
manipulates
Image
+id: string
+userId: string
+imageUrl: string
+uploadDate: Date
+analysisResult: any

Fig:3 Image Processing Package Diagram

Classes:

1. **ImageController**

- Description: Manages image-related API endpoints.
- Responsibilities: Handles image upload requests and returns analysis results.
- Operations: uploadImage(), getAnalysisResult()

2. **ImageService**
- Description: Provides business logic for image processing.
- Responsibilities: Manages image storage, preprocessing, and invokes analysis models.
- Operations: storeImage(), preprocessImage(), analyzeImage()

3. **ImageRepository**
- Description: Interacts with Azure Blob Storage or Google Firebase to store images.
- Responsibilities: Performs CRUD operations on image files.
- Operations: saveImage(), getImage()

4. **Image**
- Description: Represents the image entity.
- Attributes: id, userId, imageUrl, uploadDate, analysisResult

5.2.3 Chatbot Package

Description: This package provides the chatbot functionality for user interaction and assistance.

Classes:

1. **ChatbotController**
- Description: Manages chatbot-related API endpoints.
- Responsibilities: Handles user queries and returns responses from the chatbot.
- Operations: getChatbotResponse()

2. **ChatbotService**
- Description: Provides business logic for chatbot interactions.
- Responsibilities: Processes user inputs, interacts with NLP models, and generates responses.
- Operations: processQuery(), generateResponse()

3. **ChatbotModel**
- Description: Represents the NLP model used by the chatbot.
- Attributes: modelId, version, modelFile

5.2.4 Report Management Package

Description: This package manages the generation, storage, and retrieval of diagnostic reports.

Diagram:

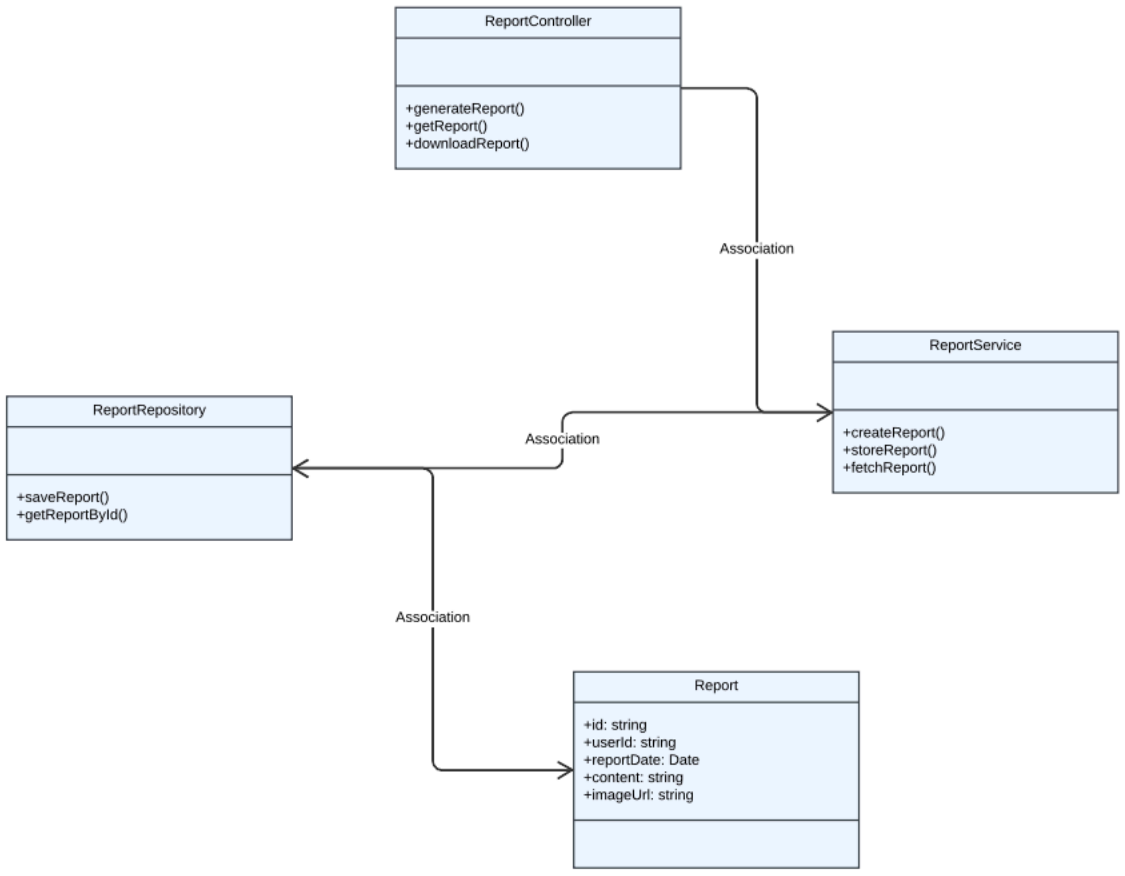


Fig:4 Report Management Package Diagram

Classes:

1. **ReportController**
- Description: Manages report-related API endpoints.
- Responsibilities: Handles requests for generating, viewing, and downloading reports.
- Operations: generateReport(), getReport(), downloadReport()

2. **ReportService**
- Description: Provides business logic for report management.
- Responsibilities: Generates diagnostic reports from analysis results and stores them.
- Operations: createReport(), storeReport(), fetchReport()

3. **ReportRepository**
- Description: Interacts with the database to store and retrieve reports.
- Responsibilities: Performs CRUD operations on report documents.
- Operations: saveReport(), getReportById()

4. **Report**
- Description: Represents the report entity.
- Attributes: id, userId, reportDate, content, imageUrl

5.2.7 Frontend Package

Description: Contains the React components for the user interface.

Classes:

1. App
- Description: The root component of the React application.
- Responsibilities: Initializes the application and routes to different views.

2. Header
- Description: Represents the header section of the application.
- Attributes: title, navigationLinks

3. LoginForm
- Description: Handles user login.
- Responsibilities: Captures user credentials and submits them for authentication.
- Operations: handleSubmit()

4. Dashboard
- Description: Provides the main user interface for logged-in users.
- Attributes: userData, recentReports

Class diagram

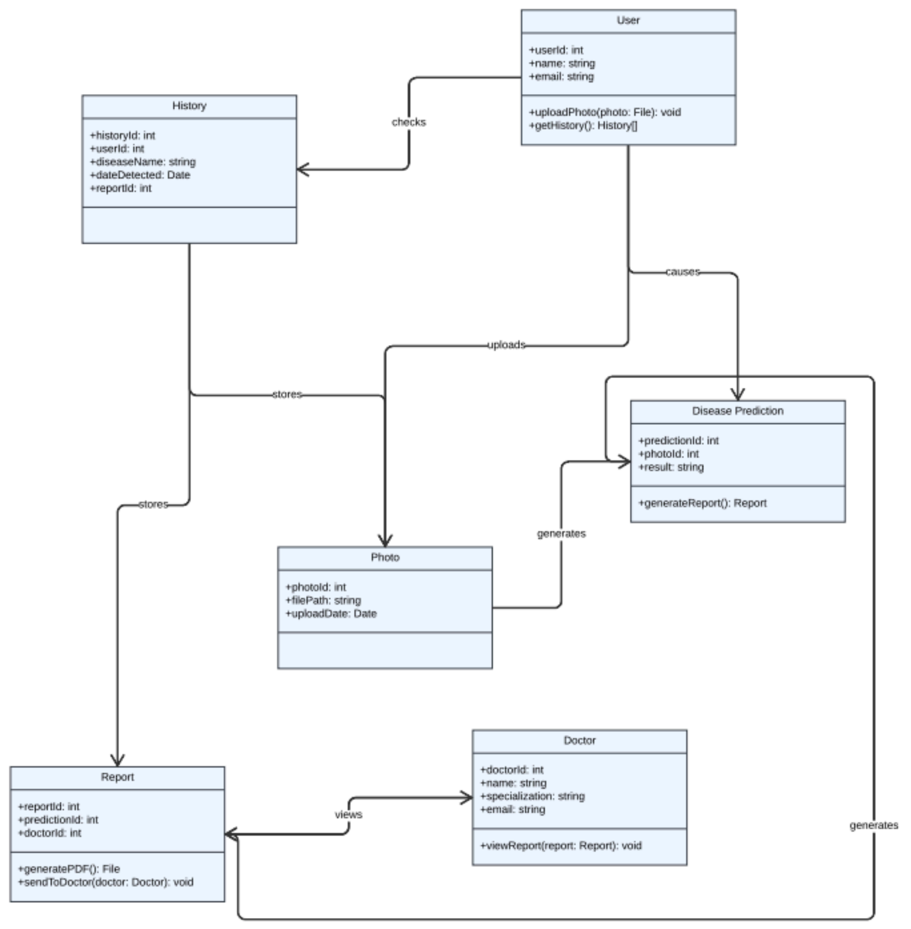


Fig 5: Frontend Package Diagram

This class diagram represents a system for managing disease predictions based on photo uploads. The central classes are `User`, `History`, `Photo`, `Disease Prediction`, `Report`, and `Doctor`.

- The `User` class has attributes for `userid`, `name`, and `email`, and methods to upload a photo and retrieve their history.
- The `History` class links a user's past disease detections, storing attributes like `historyid`, `userid`, `diseaseName`, `dateDetected`, and `reportid`.
- The `Photo` class holds the details of uploaded photos, including `photoid`, `filePath`, and `uploadDate`.

- The `Disease Prediction` class processes the uploaded photos to predict diseases, with attributes `predictionid`, `photoid`, and `result`. It also generates reports.
- The `Report` class stores the results of predictions, associating them with `reportid`, `predictionid`, and `doctorid`. It includes methods to generate PDF reports and send them to doctors.
- The `Doctor` class, characterized by `doctorid`, `name`, `specialization`, and `email`, can view reports and is responsible for analyzing them.

Relationships are defined as follows:

- A `User` uploads photos, which are stored in the `Photo` class.
- Photos are used by the `Disease Prediction` class to generate predictions.
- Predictions result in the generation of `Report` instances.
- `Doctors` view these reports and provide their analyses.
- `History` records the disease detection history of a user, which is checked by the user.

Overall, the diagram illustrates the flow from user photo upload to disease prediction, report generation, and doctor analysis, maintaining a record of disease history for each user.

6. Process View

This section describes the system's decomposition into lightweight and heavyweight processes. The HealthBot+ system consists of various interacting processes, each handling distinct aspects of the system's functionality. The main modes of communication between processes include HTTP requests/responses, WebSockets, and database queries.

6.1 Lightweight Processes

1. **User Interaction Process:**

- Components: Frontend (React), API Controller
- Function: Handles user inputs, sends HTTP requests to the backend, and displays responses.
- Communication: HTTP requests/responses

2. **Image Processing Process:**

- Components: Image Controller, Image Service, Image Repository
- Function: Manages the uploading, processing, and classification of images.
- Communication: HTTP requests, database queries

3. **Chatbot Interaction Process:**

- Components: Chatbot Controller, Chatbot Service

- Function: Manages user interactions with the chatbot, processes messages, and generates responses.
- Communication: WebSockets, internal function calls

4. **Report Generation Process:**

- Components: Report Controller, Report Service, Report Repository
- Function: Handles the creation, storage, and retrieval of reports.
- Communication: HTTP requests, database queries

6.2 Heavyweight Processes

1. **Backend Server Process:**

- Components: API Package, Database Access Package
- Function: Manages all backend operations, including user authentication, data processing, and API routing.
- Communication: HTTP requests/responses, database connections

2. **Database Process:**

- Components: MongoDBConnector, DatabaseConfig
- Function: Manages database connections and configurations.
- Communication: Database queries

3. **Frontend Process:**

- Components: React components
- Function: Provides the user interface for interacting with the system.
- Communication: HTTP requests/responses

6.3 Communication Modes

1. **HTTP Requests/Responses:**

- Used for communication between the frontend and backend processes.
- Handles user requests and sends back responses.

2. **Database Queries:**

- Used for data retrieval and storage between the backend and database processes.

3. **WebSockets:**

- Used for real-time communication between the frontend and the chatbot process.

6.4 Activity Diagram

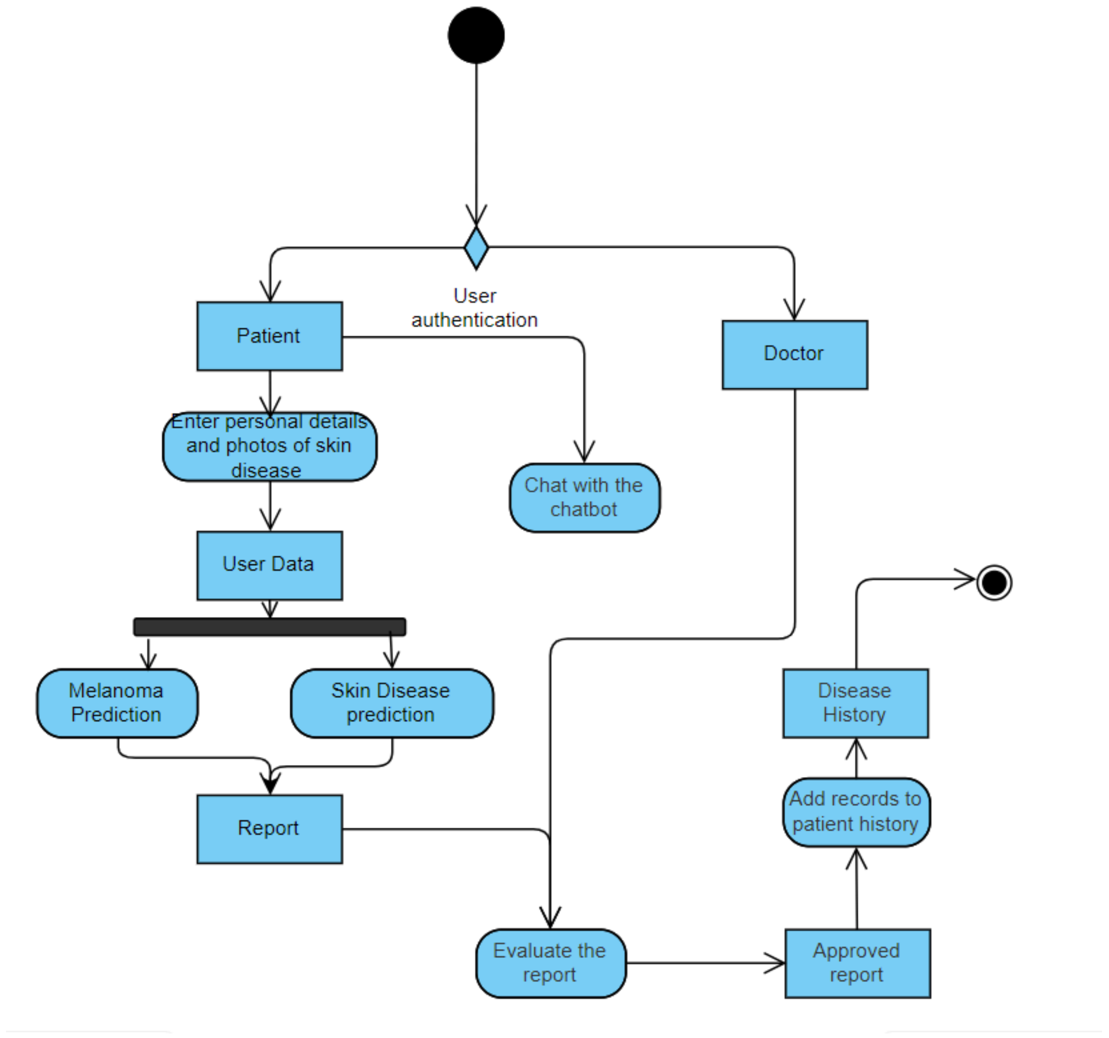


Fig:6 Activity Diagram

Description:

1. User Authentication: The process starts with user authentication, determining whether the user is a patient or a doctor.

2. Patient Workflow: If the user is a patient, they enter their personal details and upload photos of their skin condition.
The system collects the user data, which is then branched into two prediction processes:

- Melanoma Prediction: The system performs a melanoma prediction based on the uploaded photos.
- Skin Disease Prediction: The system also predicts other skin diseases using the

provided data.
- The results from these predictions are compiled into a report.

3. Doctor Workflow: If the user is a doctor, they can chat with a chatbot, possibly for assistance or additional information.
- Doctors evaluate the generated reports from the prediction processes.
- Upon evaluation, the doctor approves the report.
- The approved report is then used to update the patient's disease history, adding records to the patient history database.

4. Completion: The process concludes with the disease history being updated for future reference and further medical consultation if needed.

6.5 Sequence Diagram

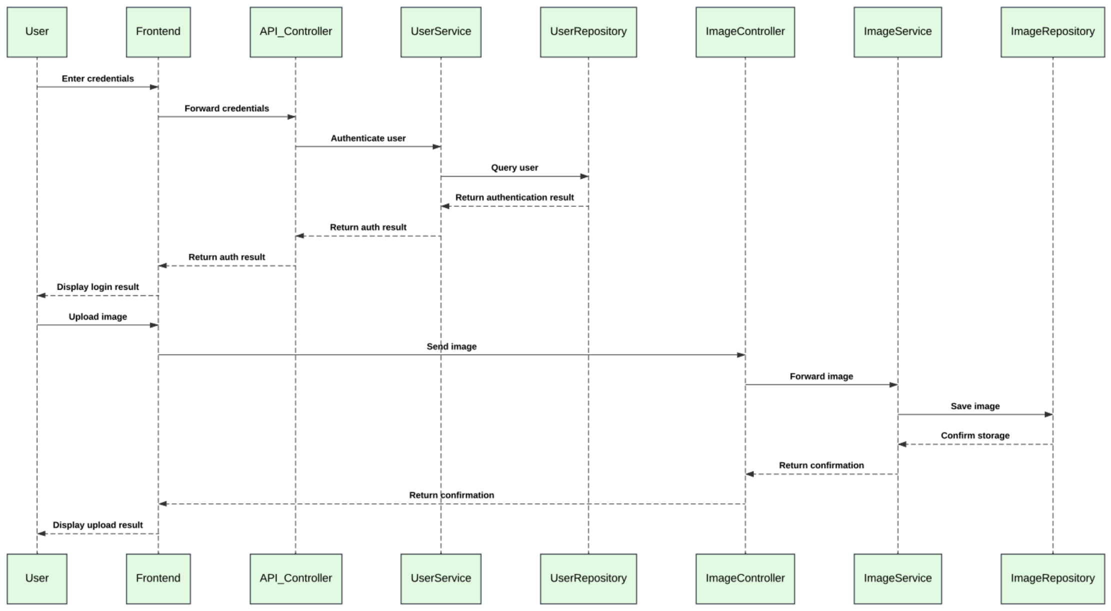


Fig:7 Sequence Diagram

Description:

1. **Login Sequence:**
    - User sends login credentials to the frontend.
    - Frontend sends credentials to the API controller.
    - API controller forwards the request to the UserService.
    - UserService interacts with the UserRepository to authenticate the user.

- UserRepository returns the authentication result to the UserService.
- UserService sends the result back to the API controller.
- API controller sends the response back to the frontend.
- Frontend updates the UI based on the login result.

2. **Image Upload Sequence:**
    - User uploads an image through the frontend.
    - Frontend sends the image to the ImageController.
    - ImageController forwards the image to the ImageService.
    - ImageService processes the image and stores it using ImageRepository.
    - ImageRepository saves the image and returns a confirmation.
    - ImageService sends the confirmation back to the ImageController.
    - ImageController sends the response back to the frontend.
    - Frontend updates the UI with the upload confirmation.

7. Deployment View

The Deployment View of the HealthBot+ system outlines the physical network configuration on which the software is deployed and executed. This view describes the physical nodes, their interconnections, and the mapping of processes onto these nodes.

7.1 Physical Network Configuration

The HealthBot+ system is deployed on a cloud infrastructure to ensure scalability, reliability, and accessibility. The primary components of the physical network configuration are:

Client Devices:

- **Description:** User devices such as personal computers, smartphones, and tablets.
- **Role:** Used by end-users (patients and doctors) to interact with the HealthBot+ system via a web browser or mobile application.
- **Connection:** Internet

Frontend Server:

- **Description:** A cloud-hosted server responsible for serving the frontend React application.
- **Role:** Delivers the user interface to client devices.
- **Connection:** Internet, communicates with Backend Server through HTTP/HTTPS.

Backend Server:

- **Description:** A cloud-hosted server running the Flask application.
- **Role:** Handles API requests, business logic, and image processing.
- **Connection:** Internet, communicates with the Database Server and Storage Server through HTTP/HTTPS and internal network protocols.

Database Server:

- **Description:** A cloud-hosted MongoDB instance.
- **Role:** Stores user data, image metadata, disease predictions, and reports.
- **Connection:** Internal network, communicates with the Backend Server.

Storage Server:

- **Description:** Cloud storage service (e.g., Azure Blob Storage or Google Firebase Storage).
- **Role:** Stores uploaded images and generated reports.
- **Connection:** Internal network, communicates with the Backend Server.
-

Machine Learning Model Server:

- **Description:** A cloud-hosted server running TensorFlow/Keras and Hugging Face models.
- **Role:** Performs image classification and disease prediction.
- **Connection:** Internal network, communicates with the Backend Server.

7.2 Process Mapping

The following mapping describes how the processes from the Process View are deployed onto the physical nodes:

User Interaction Process:

Nodes: Client Devices, Frontend Server, Backend Server

Description: Users interact with the system through their devices, which communicate with the Frontend Server to fetch and display the user interface. The Frontend Server forwards user requests to the Backend Server for processing.

Image Processing Process:

Nodes: Client Devices, Backend Server, Storage Server, Machine Learning Model Server

Description: Users upload images via client devices, which are sent to the Backend Server. The Backend Server processes and stores images on the Storage Server and performs classification using the Machine Learning Model Server.

Chatbot Interaction Process:

Nodes: Client Devices, Backend Server

Description: Users interact with the chatbot through their devices. The Backend Server processes messages and generates responses in real-time.

Report Generation Process:

Nodes: Backend Server, Storage Server, Database Server

Description: The Backend Server generates reports based on disease predictions, stores them on the Storage Server, and updates the Database Server with report metadata.

Database Access Process:

Nodes: Backend Server, Database Server

Description: The Backend Server queries and updates data on the Database Server as required by different processes.

8. Implementation View

This section provides a detailed description of the implementation model of the skin disease diagnosis web app. The web app is structured using the Model-View-Controller (MVC) architectural pattern, which separates the application into three interconnected components, allowing for efficient code management and scalability.

8.1 Overview

The skin disease diagnosis web app is divided into three main layers: Model, View, and Controller. Each layer has specific responsibilities and interacts with the others to deliver the functionality of the application.

Model Layer: This layer handles all the data-related logic. It includes the machine learning and deep learning models for skin disease diagnosis, the user data models, and the report generation models.

View Layer: This layer is responsible for the presentation of the data. It includes the user interface components such as HTML, CSS, and JavaScript files that render the web pages.

Controller Layer: This layer acts as an intermediary between the Model and View layers. It processes user inputs, invokes model updates, and determines which views to render.

A component diagram representing the relations between the layers is included below:

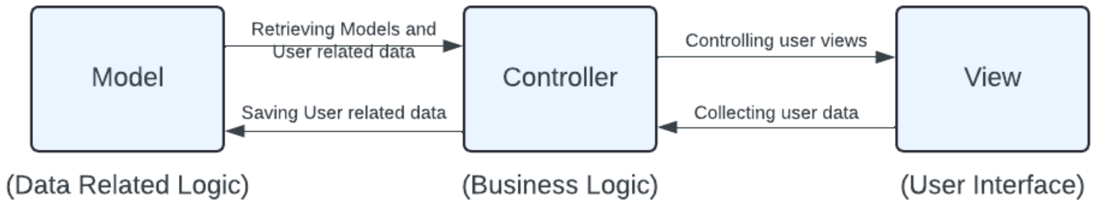


Fig:8 component diagram representing the relations between the layers

8.2 Layers

8.2.1 Model Layer

The Model layer includes the following subsystems:

- **User Management Subsystem**: Handles user authentication and authorization.
- **Disease Diagnosis Subsystem**: Contains machine learning and deep learning models for diagnosing skin diseases.
- **Report Generation Subsystem:** Generates medical reports based on the diagnosis results.

Component Diagram for Model Layer:

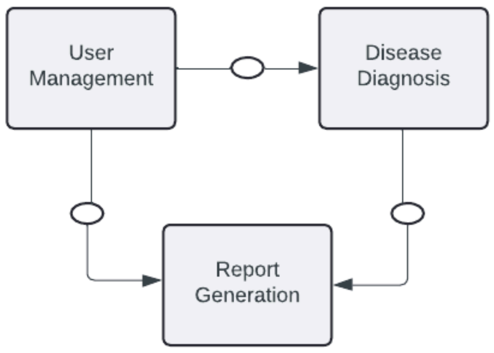


Fig: 9 Component Diagram for Model Layer

8.2.2 View Layer

The View layer includes the following subsystems:

- **Login Subsystem:** Manages user login and registration interfaces.
- **Image Upload Subsystem:** Provides the interface for users to upload images of skin diseases.
- **Result Display Subsystem:** Shows the diagnosis results to the user.
- **Report Display Subsystem:** Displays the generated medical reports.
- **Chatbot Interface Subsystem:** Provides the interface for the chatbot interactions.

Component Diagram for View Layer:

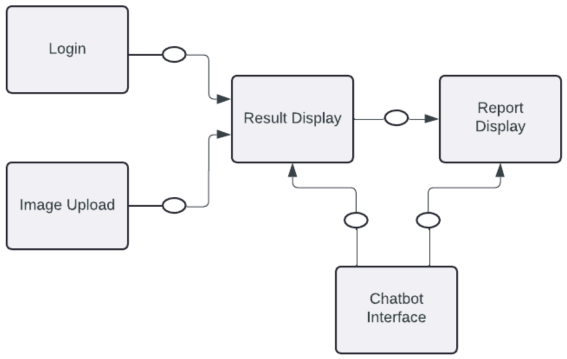


Fig:10 Component Diagram for View Layer

8.2.3 Controller Layer

The Controller layer includes the following subsystems:

- **Login Controller:** Manages the logic for user authentication and redirects.
- **Image Processing Controller:** Handles the image upload and processing logic.
- **Diagnosis Controller:** Coordinates the interaction between the uploaded image and the diagnosis models.
- **Report Controller:** Manages the report generation and retrieval logic.
- **Chatbot Controller:** Handles the interactions between the user and the chatbot.

Component Diagram for Controller Layer:

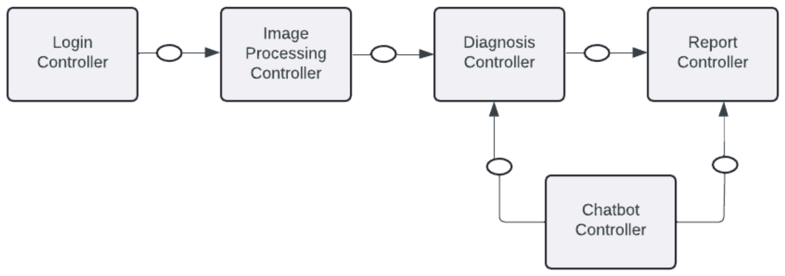


Fig 11: Component Diagram for Controller Layer

Package Diagram

A package diagram illustrating the overall structure and dependencies of the various subsystems within each layer is included below:

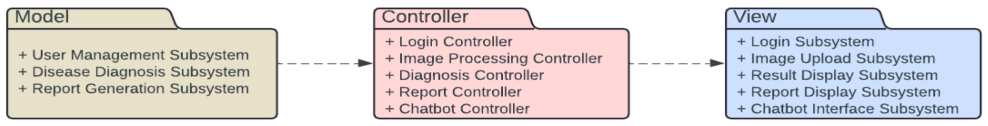

Model
+ User Management Subsystem
+ Disease Diagnosis Subsystem
+ Report Generation Subsystem
Controller
+ Login Controller
+ Image Processing Controller
+ Diagnosis Controller
+ Report Controller
+ Chatbot Controller
View
+ Login Subsystem
+ Image Upload Subsystem
+ Result Display Subsystem
+ Report Display Subsystem
+ Chatbot Interface Subsystem

Fig: 12 Package Diagram

The Model Layer package includes sub-packages for User Management, Disease Diagnosis, and Report Generation. These sub-packages contain classes and interfaces that manage data and business logic.

The View Layer package consists of sub-packages for Login, Image Upload, Result Display, Report Display, and Chatbot Interface, which contain the user interface components.

The Controller Layer package includes sub-packages for Login Controller, Image Processing Controller, Diagnosis Controller, Report Controller, and Chatbot Controller, which contain the control logic for managing user interactions and model updates.

9. Size and Performance

The skin disease diagnosis web app is designed to handle a variety of tasks, including user authentication, image processing, machine learning model inference, and report generation. To ensure optimal performance and scalability, the architecture has been designed with the following dimensioning characteristics and target performance constraints:

1. **Image Upload and Processing**: The web app supports uploading images up to 5 MB in size. Images are processed and resized on the server to ensure uniformity and to optimize the performance of the machine learning models.
2. **Machine Learning Model Inference**: The inference time for the machine learning and deep learning models is optimized to provide diagnosis results within a few seconds. This is achieved by utilizing pre-trained models and optimizing them for fast inference using techniques like model quantization.
3. **Concurrent Users**: The system is designed to support up to 100 concurrent users, ensuring that multiple users can upload images and receive diagnosis results simultaneously without significant performance degradation.
4. **Report Generation**: Medical reports are generated in PDF format and stored on the server. The system is optimized to generate and retrieve reports quickly, with a target of generating reports within a few seconds (approximately, under 5).
5. **Response Time:** The target response time for user interactions, such as login, image upload, and chatbot queries, is within a few seconds (approximately, under 3). This ensures a smooth and responsive user experience.
6. **Scalability**: The architecture is designed to scale horizontally by adding more servers or instances to handle increased load. This is facilitated by using containerization and orchestration tools like Docker and Kubernetes.

10. Quality

1. **Extensibility:** The use of the MVC architecture promotes extensibility by separating concerns. New features or updates can be added to the Model, View, or Controller layers independently without affecting the other layers. For instance, new machine learning models or additional functionality in the user interface can be integrated with minimal changes to the existing codebase.
2. **Reliability:** The system ensures reliability through error handling, logging, and monitoring. The architecture includes redundancy for critical components, such as load balancers and databases, to prevent single points of failure. Regular backups and failover mechanisms are implemented to maintain data integrity and availability.
3. **Portability:** The web app is developed using platform-independent technologies and frameworks, ensuring that it can be deployed on various operating systems and cloud platforms. The use of containerization with Docker allows for easy deployment and migration across different environments.

4. **Security:** The architecture incorporates security best practices, including user authentication and authorization, secure communication using HTTPS, and protection against common web vulnerabilities like SQL injection and cross-site scripting (XSS). Sensitive data, such as user credentials and medical information, are encrypted both in transit and at rest.
5. **Privacy:** User privacy is maintained by implementing data anonymization and access control measures. Only authorized personnel have access to sensitive data, and users have control over their data through account settings and privacy options.
6. **Maintainability:** The codebase is organized and documented following industry best practices, making it easier for developers to understand, maintain, and extend the system. Automated testing and continuous integration/continuous deployment (CI/CD) pipelines are used to ensure code quality and facilitate rapid development cycles.
7. **Performance Optimization:** The architecture includes performance monitoring and optimization techniques, such as caching frequently accessed data, optimizing database queries, and using content delivery networks (CDNs) for static assets. Regular performance testing and profiling help identify and address bottlenecks.