**“HealthBot : AI for Early Disease Detection”**

***A***

***Project Report***

*submitted in partial fulfillment of the*

*requirements for the completion of Internship at*

**

**in**

**BACHELOR OF TECHNOLOGY**

**COMPUTER SCIENCE & ENGINEERING**

**by**

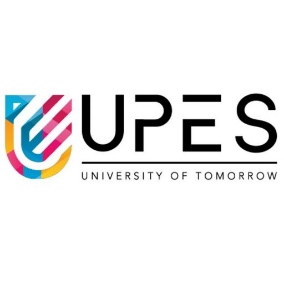
***under the guidance of***

**Ms.Pooja**

**(IBM Mentor)**

**Mr. Kotha Venugopalachary**

**(Internal Mentor)**



**School of Computer Science**

**University of Petroleum & Energy Studies**

**Bidholi, Via Prem Nagar, Dehradun, Uttarakhand**

**CANDIDATE’S DECLARATION**

We hereby certify that the project work entitled **“HealthBot”** in partial fulfilment of the requirements for the completion of IBM Internship and award of the Degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with specialization in Artificial Intelligence and Machine Learning and submitted to the Department of Systemics, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of my/ our work carried out during a period from **June**, **2025** to **August**, **2025** under the supervision of  **Prof. Kotha Venugopalachary** and **Ms. Pooja.**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Date:** 30/07/2025  **Mr. Kotha Venugopalachary**

(Professor)

**ABSTRACT**

The Early Disease Detection AI is an intelligent, interactive system that is built to help users determine possible health issues from their symptoms. The project combines a machine learning-driven backend with a simple to use web frontend to provide real-time, tailored diagnostic recommendations. Symptoms and the time sick may be input through a conversational chatbot interface, which simulates a natural diagnostic process by asking follow-up questions based on probable conditions.

The heart of the system is a trained decision tree model that processes binary-encoded symptom data to make high-interpretability predictions for potential diseases. The model is complemented by support files that hold symptom descriptions, severity scores, and precautionary advice. The backend, implemented using Flask, provides a RESTful API that accepts user input, makes predictions about conditions, estimates severity from symptom duration, and provides health advice in a structured representation.

The frontend, built with React and Tailwind CSS, provides fluid navigation and user experience on multiple devices. It enables users to initiate a diagnosis, see results such as the name of the disease, calculate confidence score, and precautions, and comprehend the gravity of their situation. The structure prioritizes modularity, scalability, and cross-platform support.

Although the system gives informative diagnoses and actionable suggestions, it states clearly that it's for educational purpose only and not to be used as a substitute for a medical professional's advice. With additional refinement, i.e., voice-entry symptoms, image-based diagnostics support, or integration with medical libraries, this tool has immense potential to aid telemedicine and early intervention methods, particularly in low-resource or remote environments.

**TABLE OF CONTENTS**

**1. Introduction**  
1.1. Problem Statement  
1.2. Project Timeline and PERT Chart

**2. System Analysis**  
2.1. System Objectives and Scope  
2.2. Need for Early Disease Detection  
2.3. Analysis of Current Gaps in Healthcare Access

**3. System Architecture and Design**  
3.1. High-Level System Architecture  
3.2. Design Components – Frontend, Backend, and ML Engine

**4. Use Case and Design Diagrams**  
4.1. Use Case Diagram  
4.2. Class Diagram  
4.3. Sequence Diagram  
4.4. Data Flow Diagram (Level 0)

**5. Technology Stack**  
5.1. Frontend: React.js with Tailwind CSS  
5.2. Backend: Flask and Python  
5.3. Machine Learning: scikit-learn, pandas, NumPy  
5.4. Tools: VS Code, Git, Postman

**6. Dataset, Preprocessing, and EDA**  
6.1. Dataset Overview  
6.2. Feature Engineering and Label Encoding  
6.3. Data Cleaning and Vector Preparation  
6.4. Exploratory Data Analysis Insights

**7. Machine Learning Engine**  
7.1. Model Selection and Justification  
7.2. Model Training and Saving  
7.3. Confidence Scoring and Severity Assessment  
7.4. Integration with Chatbot Logic

**8. Implementation Details**  
8.1. System Structure Overview  
8.2. Frontend–Backend Communication Flow  
8.3. Flask API and ML Inference Logic  
8.4. Real-Time Diagnosis Pipeline

**9. Output Screens**  
9.1. Landing Page  
9.2. Symptom Input Interface  
9.3. Chat Interaction Preview  
9.4. Final Diagnosis and Recommendations

**10. Evaluation and Results**  
10.1. Model Accuracy and Metrics  
10.2. Cross-Validation Results  
10.3. Real Input–Output Examples  
10.4. Observations and Reliability

**11. Limitations and Future Enhancements**  
11.1. Current Limitations  
11.2. Proposed Future Improvements

**12. Conclusion**

**References**

1. **Introduction**

1. Introduction

Over the last few years, the incorporation of artificial intelligence (AI) in the field of healthcare has exhibited vast potential for enhancing the detection of diseases at early stages, particularly in resource-scarce environments. Early detection of diseases can drastically enhance treatment as well as lessen the burden on healthcare expenditure. Nevertheless, poor access to medical specialists, inadequate awareness, and delayed symptom realization do frequently impede early diagnosis—mainly in rural or underprivileged areas.

The initiative, named AI for Early Disease Detection, seeks to fill this gap by providing users with a virtual, AI-based assistant that assists them in assessing their symptoms and getting preliminary diagnostic information. Users can simply chat with a bot, provide their symptoms and duration of sickness, and the system will infer the most likely disease, infer severity, offer a condition description, and suggest precautions.

The system uses a machine learning pipeline that is trained on an exhaustive symptom-disease dataset. A decision tree model, selected due to its explainability, predicts symptom patterns to potential diseases. Prediction logic and data processing are handled by the backend, developed with Flask. The frontend, using React and styled with Tailwind CSS, provides an easy-to-use and responsive user interface.

This product is not designed to substitute for medical care, but rather act as an initial aid to symptom analysis and education. Its modular architecture and web-based interface render it usable, scalable, and compatible with potential integration into larger healthcare applications. The project illustrates the ability of AI to aid in proactive health surveillance and inform users about their symptoms so that they can receive appropriate medical care in a timely manner.

1.1 Problem Statement

Even with advances in health care, early detection of disease is one of the biggest challenges—especially in rural and underserved communities where there is limited access to medical practitioners. People do not realize the severity of the symptoms because they lack information or medical advice, thus leading to late diagnoses and aggravated health status. Conventional health systems become bogged down and cannot expand to accommodate immediate consultations for everyone.

There is an urgent need for a solution that is accessible, scalable, and smart enough to help users analyze symptoms early and with certainty. The system should offer initial health feedback, indicate potential conditions, and prompt timely consultation with doctors, particularly in regions where proper healthcare infrastructure is limited.

The solution proposed, AI for Early Disease Detection, seeks to fill this divide by providing a virtual AI assistant who can take into account user-reported symptoms, deduce likely diseases, summarize their severity and nature, and provide precautionary tips. Through the use of machine learning and an easy-to-use web interface, the system allows users to take proactive action towards their health while minimizing reliance on instant medical access for simple symptom assessment.

1.2 Project Timeline and PERT Chart Legend

1. Preliminary Research

* 1.1 Literature Survey
  + Study of disease detection using AI
  + Review of existing chatbot health advisors
* 1.2 Dataset Exploration
  + Research various medical datasets (e.g., skin, eye, general symptoms)
  + Shortlist datasets for experimentation

2. Experimental Mini-Projects

* 2.1 Skin Cancer Detection
  + Dataset: ISIC (or similar)
  + Model: CNN-based classifier
  + Outcome: Functional prototype, insights into image-based diagnosis
* 2.2 Diabetic Retinopathy Detection
  + Dataset: EyePACS (or similar)
  + Model: InceptionNet or ResNet-based
  + Outcome: Learned about preprocessing fundus images and patient metadata
* 2.3 Roadblock
  + Realization: These datasets don't align well with our vision of an interactive chatbot for general disease detection

3. Final Dataset Identification

* 3.1 Dataset Acquisition
  + Identified a structured symptom-to-disease dataset (e.g., Training.csv with 132 symptoms and 41 diseases)
  + Validated quality and completeness of data
* 3.2 Data Preprocessing
  + Cleaned, encoded, and prepared the dataset
  + Created feature vectors for training

4. Model Development

* 4.1 Model Selection
  + Chose Decision Tree for interpretability
* 4.2 Model Training
  + Trained model on cleaned dataset
  + Used label encoders and feature importance
* 4.3 Evaluation
  + Assessed accuracy, precision, recall
  + Saved best model and encoders

5. Backend Development

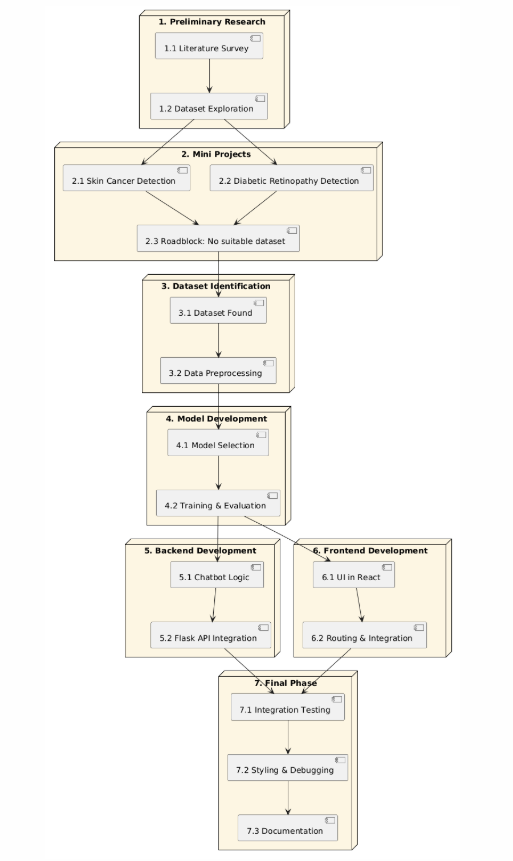
* 5.1 Chatbot Logic
  + Created HealthcareChatbot class to handle prediction, severity, precautions
* 5.2 API Integration
  + Built Flask API (app.py) to serve frontend requests

6. Frontend Development

* 6.1 UI Design
  + Designed chat interface using React + Tailwind CSS
* 6.2 Routing
  + Added navigation (Home, Chat, NotFound) via React Router
* 6.3 Integration
  + Connected frontend to Flask API for real-time diagnosis

7. Final Deployment & Testing

* 7.1 Integration Testing
  + Verified end-to-end flow
* 7.2 UI Polish and Styling
* 7.3 Debugging and Improvements
* 7.4 Documentation and Presentation



**2. System Analysis**

**1. Overview**

The AI for Early Disease Detection system is designed as a multi-phase health advisory tool that uses machine learning to analyze user symptoms and predict possible diseases. It provides a preliminary diagnosis, assesses condition severity, and recommends preventive measures. The system is structured to be accessible, interactive, and informative, particularly for users who may not have immediate access to medical professionals.

**2. Existing System**

Traditional disease diagnosis systems often require:

* In-person consultations with doctors.
* Manual symptom evaluation.
* Specialized datasets for specific diseases (e.g., cancer, retinopathy).
* Lack of interactive or intelligent frontends for non-technical users.

Limitations:

* Limited availability in rural/remote areas.
* High dependency on professional interpretation.
* No real-time interaction or feedback.
* Specialized systems often focus only on a narrow range of diseases.

**3. Proposed System**

The proposed system overcomes these limitations by:

* Offering a symptom-based chatbot interface for diagnosis.
* Using a trained decision tree classifier to predict general diseases.
* Providing additional features like severity assessment and precaution suggestions.
* Offering a responsive, web-based frontend that can be accessed on any device.

**Key Modules:**

* Symptom Input Module: User interacts via chatbot or form to provide symptoms and illness duration.
* Prediction Engine: Uses a decision tree model trained on a labeled dataset to classify diseases.
* Severity Assessment: Calculates risk based on symptom severity values and duration.
* Information Module: Provides disease descriptions and precautionary advice.
* Frontend Interface: Built using React for seamless user interaction.
* API Layer: Flask backend to connect frontend and ML model.

**4. Feasibility Analysis**

**a. Technical Feasibility**

* Uses standard and proven technologies (Python, Scikit-learn, React, Flask).
* Easily scalable and adaptable to future enhancements like voice input or multilingual support.

**b. Operational Feasibility**

* Requires only basic symptom inputs from the user.
* Minimal technical knowledge needed to operate the system.
* Accessible from any modern web browser.

**c. Economic Feasibility**

* Uses publicly available datasets and open-source libraries.
* No external hardware or licensed software needed.
* Cost-effective for wide deployment.

**5. Strengths of the System**

* Covers a broad range of diseases instead of a single condition.
* High interpretability using decision tree models.
* Extensible design allows future integration with images, sensors, or user profiles.
* Real-time, personalized interaction via chatbot.

**6. Limitations**

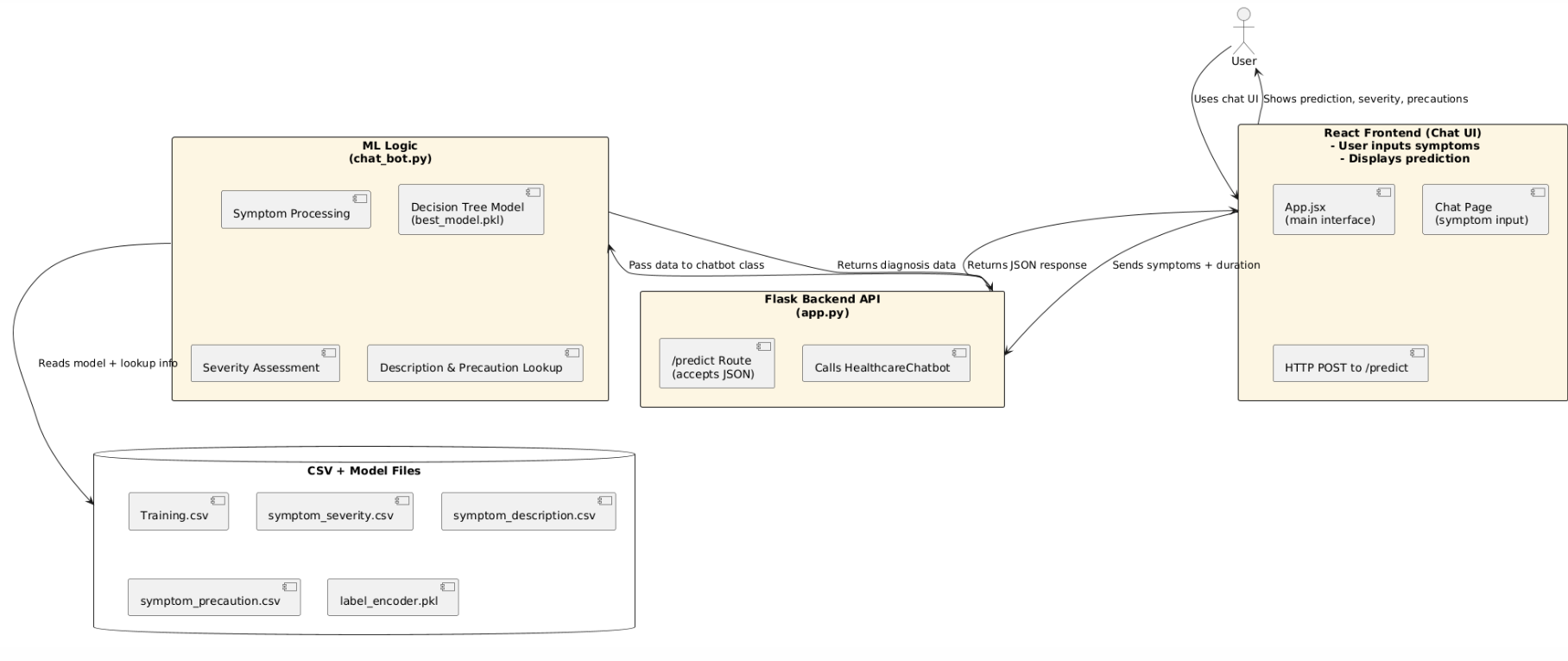
* Limited by the quality and coverage of the training dataset.
* Not a substitute for medical diagnosis by certified professionals.
* Accuracy may decrease with vague or incomplete user input.
* Cannot handle emergency or critical health conditions.

**7. Future Scope**

* Integration with speech-to-text for symptom input via voice.
* Adding multilingual support for regional accessibility.
* Incorporation of wearable device data (e.g., heart rate, temperature).
* Real-time consultation linkage with certified health professionals.
* Enhanced AI models using ensemble learning or transformers for richer prediction.

**3. System Architecture and Design**

**1. High-Level Architecture**



**2. Component Description**

a. User Interface (Frontend – React + Tailwind)

* A clean, responsive web interface for users to enter symptoms and view results.
* Routes include:
  + / – Home screen.
  + /chat – Chat interface.
* Uses React Context to manage global state.
* Sends symptom data to the Flask API and displays prediction output.

b. Backend Server (Flask)

* Handles HTTP POST requests from the frontend (/predict route).
* Passes symptom list and duration to the HealthcareChatbot class.
* Returns disease name, confidence score, severity message, description, and precautions as JSON.

c. Machine Learning Module

* HealthcareChatbot class loads:
  + Trained decision tree model (best\_model.pkl)
  + Label encoder and feature name list
  + Symptom-related CSV files: descriptions, severity, precautions
* Predicts disease based on binary vector of input symptoms.
* Uses predict\_proba (if available) for confidence scoring.
* Calculates severity based on total symptom weight × duration.

d. Datasets & Models

* Training Dataset: CSV file mapping symptoms (132 features) to 41 diseases.
* Supporting Files:
  + symptom\_description.csv
  + symptom\_severity.csv
  + symptom\_precaution.csv
* Model Files:
  + best\_model.pkl
  + label\_encoder.pkl
  + feature\_names.pkl

**3. Design Patterns Used**

* Model-View-Controller (MVC) separation:
  + View: React Frontend
  + Controller: Flask routes
  + Model: ML logic and prediction class
* Service Layer Abstraction: HealthcareChatbot encapsulates all model logic and keeps Flask endpoints clean.

**4. Sequence Flow**

1. User enters symptoms → submits form.
2. React frontend sends data to /predict API via POST.
3. Flask backend calls chatbot.predict\_disease() and calculate\_condition\_severity().
4. Chatbot class:
   * Encodes symptoms.
   * Predicts disease using the trained model.
   * Looks up description, severity, and precautions.
5. API returns structured JSON response.
6. Frontend displays diagnosis, advice, and health tips.

**5. Scalability & Extensibility**

* Easily extendable with:
  + Multilingual frontend
  + Image-based models (e.g., skin/eye conditions)
  + Integration with EHR (Electronic Health Records)
* Scalable as a microservice or container (Docker-ready)
* Model can be upgraded to XGBoost, Ensemble, or Transformer-based approaches in future

**4. Use Case and Design Diagrams**

**Brief Description:**

Symptoms and illness duration are input by the user via a web interface. The input is processed, a likely disease is predicted by a trained ML model, the severity is evaluated, and a description and precautionary notice are returned.

**Preconditions:**

* The web application is running and accessible.
* The Flask backend and ML model are properly loaded.
* The required symptom and precaution datasets are available.

**Postconditions:**

* The user receives a disease prediction.
* The user is informed about severity and advised on precautions.

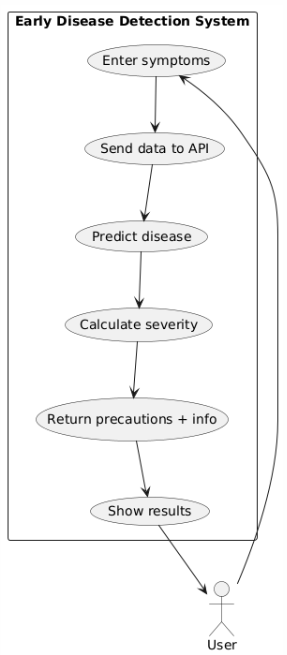
**Main Flow (Basic Path):**

| **Step** | **Actor/System** | **Action** |
| --- | --- | --- |
| 1 | User | Opens the web application and navigates to the chat interface. |
| 2 | User | Inputs primary symptom(s) and selects illness duration. |
| 3 | System (Frontend) | Sends the input to the Flask API (/predict). |
| 4 | System (Backend) | Calls HealthcareChatbot with received symptoms and duration. |
| 5 | Chatbot Logic | Matches symptoms, encodes input vector, predicts disease. |
| 6 | Chatbot Logic | Calculates severity and retrieves description + precautions. |
| 7 | System (Backend) | Returns disease name, confidence, severity, precautions to frontend. |
| 8 | System (Frontend) | Displays results to the user in a readable format. |

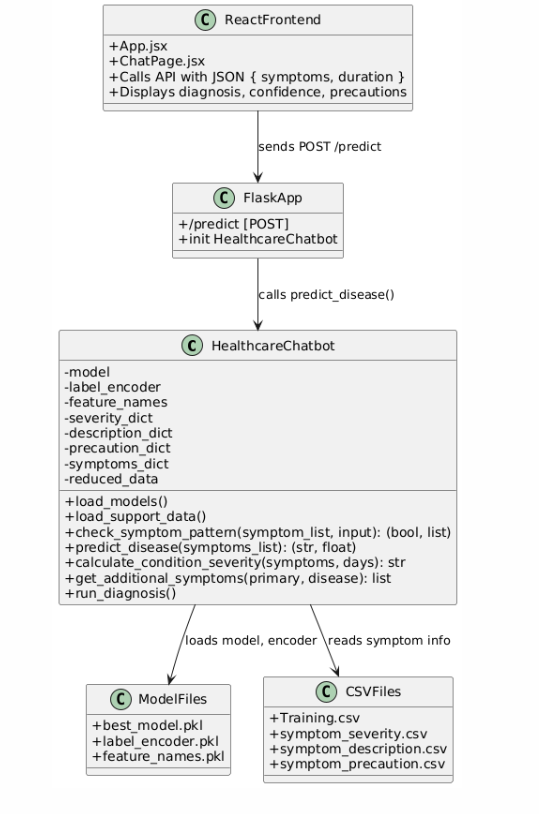
**Alternate Flows:**

* A1: No symptoms entered
  + System returns error response: “No symptoms provided.”
  + User is prompted to enter at least one symptom.
* A2: Invalid or rare symptom
  + System cannot match the symptom.
  + Prompts user to re-enter or select from suggestions.
* A3: Backend not available
  + Returns server error or fallback message: “Service temporarily unavailable.”

**4.1. Use Case Diagrams**

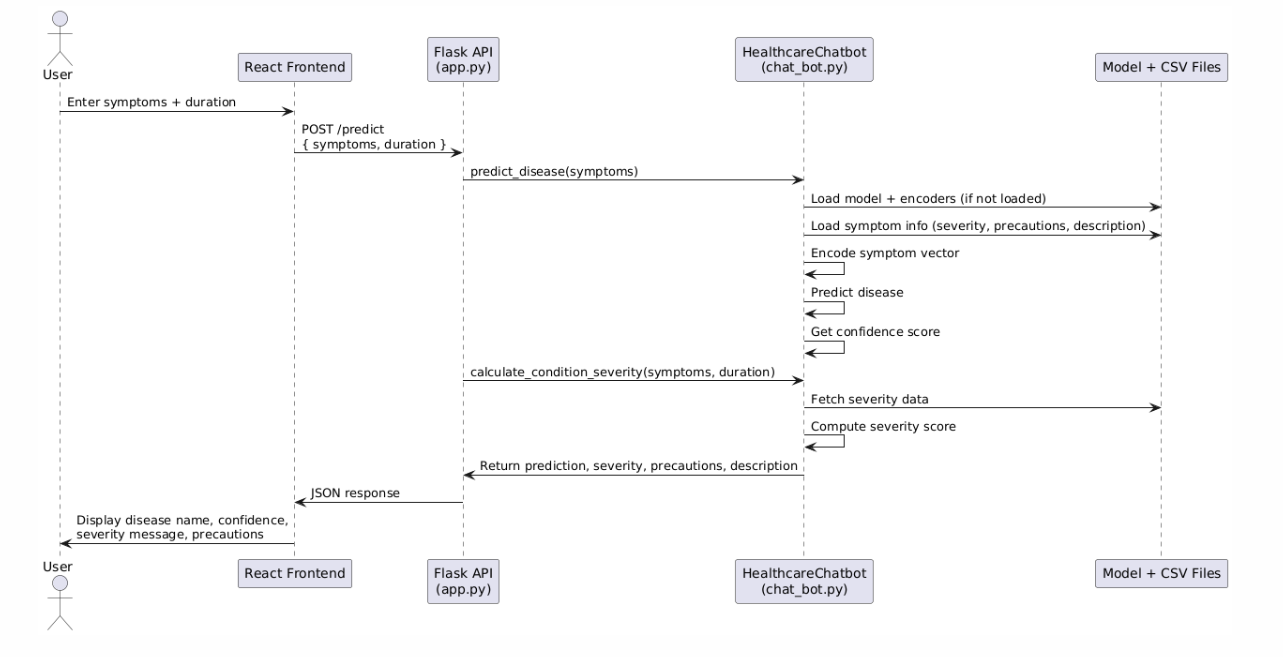


**4.2. Class Diagram:**



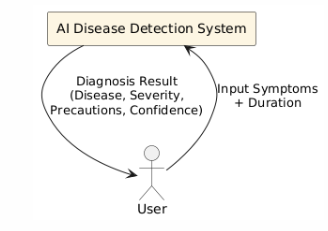
**4.3. Sequence Diagrams**

Sequence diagrams represent the dynamic flow of interactions between objects over time for a specific functionality.

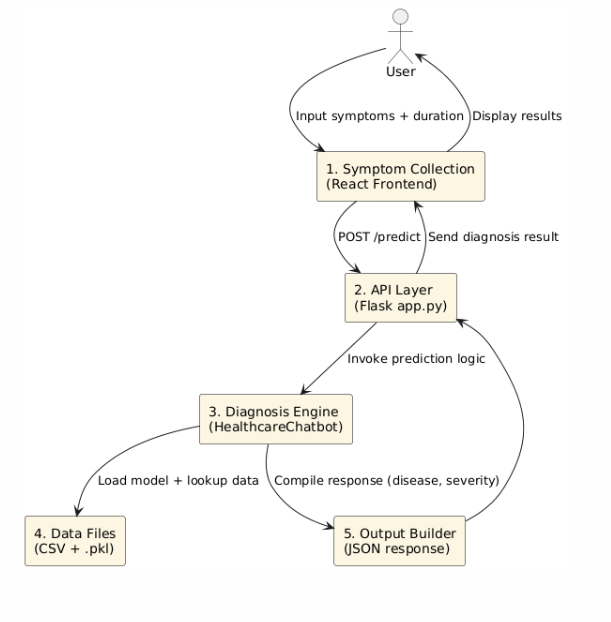


**4.5. Data Flow Diagrams (DFD):**

**Level 0 (Context Diagram):**



**Level 1:**



**5. Technology Stack**

**1. Dataset and Preprocessing**

* Kaggle Dataset: The itachi9604 "Disease Symptom Description Dataset" provides structured binary data linking ~132 symptoms to ~41 disease labels, along with textual descriptions and precaution metadata
* Data Processing Tools:
  + Pandas: Ingests CSV files like Training.csv, symptom\_severity.csv, symptom\_description.csv, and symptom\_precaution.csv.
  + NumPy: Manages symptom encoding as fixed-size binary vectors.
  + Missing Value Handling: Fills NaNs with zeros for missing symptom indicators.
  + Label Encoding: Structures disease names into numeric form using scikit‑learn LabelEncoder.

**2. Machine Learning Pipeline**

* Scikit-learn: Implements decision tree classifier trained on the binary symptom matrix—chosen for its explainability and ease of thresholding.
* Grid Search / Cross-Validation: Optimizes depth, splitting criteria, and pruning parameters for better generalization.
* Model Serialization:
  + best\_model.pkl: Final trained model.
  + label\_encoder.pkl
  + feature\_names.pkl: Symptom order mapping.
* Severity Score Logic: Reads weighted severity values to compute composite risk scores based on symptom presence and reported duration.

**3. Backend and Prediction Logic**

* Python 3.x & Flask:
  + Serves /predict endpoint.
  + Accepts JSON input: symptoms + duration.
  + Uses Flask‑CORS for secure cross-origin frontend requests.
* HealthcareChatbot Class:
  + Loads model and encoder once (singleton-like behavior to avoid repeat reads).
  + Encodes user symptoms into vector.
  + Calls .predict\_proba() or .predict() to get disease prediction and confidence.
  + Fetches description and precaution texts from dataset.
  + Calculates severity message using symptom weights × days.

**4. Frontend (React + Tailwind)**

* React.js SPA: Built as a clean, interactive chat interface.
  + ChatPage.jsx: Allows symptoms selection—using autocomplete/checkbox UI.
  + App.jsx: Houses global state using React Context.
  + HTTP: Sends POST to backend carrying symptom list and duration.
* Tailwind CSS: Utility-first styling ensures responsive, accessible design across devices.

**5. Support Data & Storage**

* Support CSV Files:
  + symptom\_severity.csv: Severity weights per symptom.
  + symptom\_description.csv & symptom\_precaution.csv: Supplementary information.
* Model Artifacts:
  + best\_model.pkl, label\_encoder.pkl, feature\_names.pkl.
* All static data loaded locally on server startup for performance (cached in RAM).

**6. Development Tools & Environment**

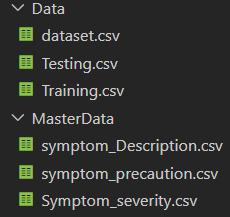
* Python Environment: Python 3.x with virtualenv or Conda.
* Node.js + npm: Manages React dependencies.
* Jupyter Notebook: Exploratory analysis and model training.
* Git: Code versioning.
* VS Code: Editor with syntax highlighting and debugging support.

**6. Data Sources and Preprocessing**

**1. Dataset Overview**

The project uses the **“Disease Symptom Description Dataset”** by *itachi9604* from Kaggle:  
 <https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset>

This dataset is a structured compilation of disease-related information, symptom mappings, textual descriptions, severity ratings, and precautionary measures. It is ideal for creating rule-based or machine learning models for disease prediction.



**Key CSV Files in the Dataset:**

| **File Name** | **Description** |
| --- | --- |
| Training.csv | Main file containing rows of diseases and binary symptom indicators (132 symptoms). |
| Testing.csv | Similar structure to Training.csv, used for model evaluation. |
| symptom\_severity.csv | Maps each symptom to a numeric severity score. |
| symptom\_description.csv | Contains textual explanations of each disease. |
| symptom\_precaution.csv | Lists four precautionary actions per disease. |

**2. Preprocessing Steps**

To convert the raw CSV files into a form usable for model training and prediction, several preprocessing steps were applied:

2.1 Data Cleaning

* Missing Values: Checked for and filled missing symptom values in Training.csv and symptom\_\*.csv files using default values (e.g., 0 for absence).
* Standardization: Converted all symptom strings to lowercase and replaced spaces with underscores to maintain consistency (e.g., “joint pain” → “joint\_pain”).
* Duplicate Removal: Ensured unique rows in training data to prevent model bias.

2.2 Symptom Encoding

* Symptoms are encoded as binary features (1 = present, 0 = absent) across 132 symptom columns.
* Each input from the user is converted into a fixed-length binary vector using the index mapping from feature\_names.pkl.

2.3 Label Encoding

* The target variable prognosis (disease name) was encoded into numeric labels using LabelEncoder from scikit-learn and stored as label\_encoder.pkl.

2.4 Train-Test Separation

* Although Testing.csv exists, custom cross-validation and model tuning were performed by splitting Training.csv into:
  + 80% training set
  + 20% validation set

2.5 Integration of Support Files

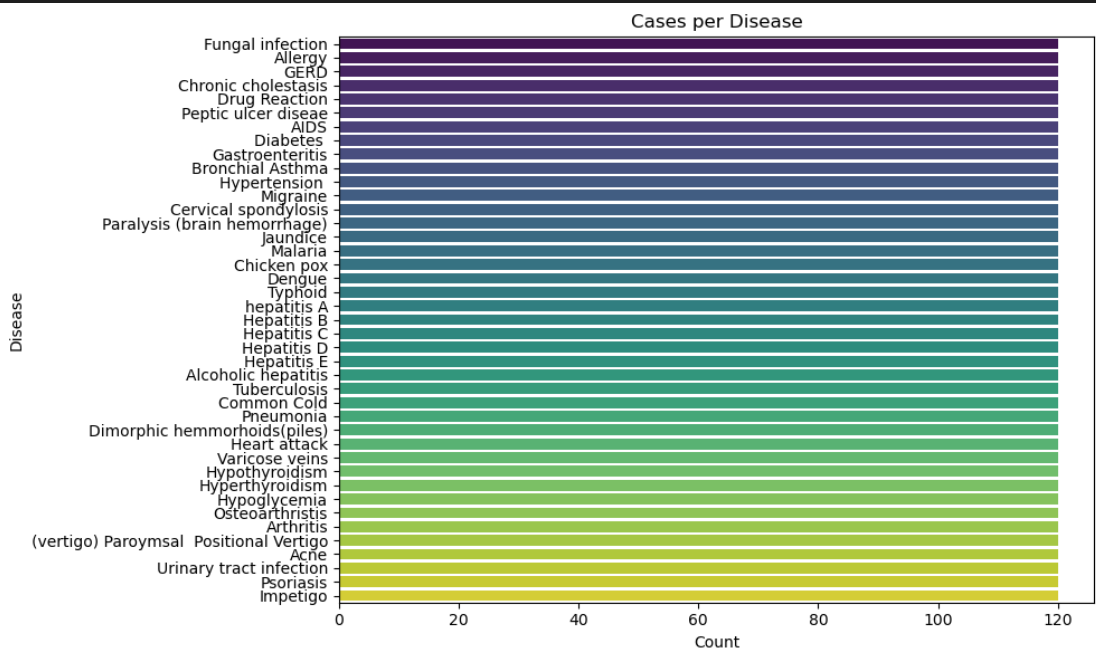
* The following dictionaries were built at runtime from supporting CSVs:
  + symptom\_severity.csv → severity\_dict
  + symptom\_description.csv → description\_dict
  + symptom\_precaution.csv → precaution\_dict
* These enrich predictions with relevant information during inference.

**3. Exploratory Data Analysis (EDA)**

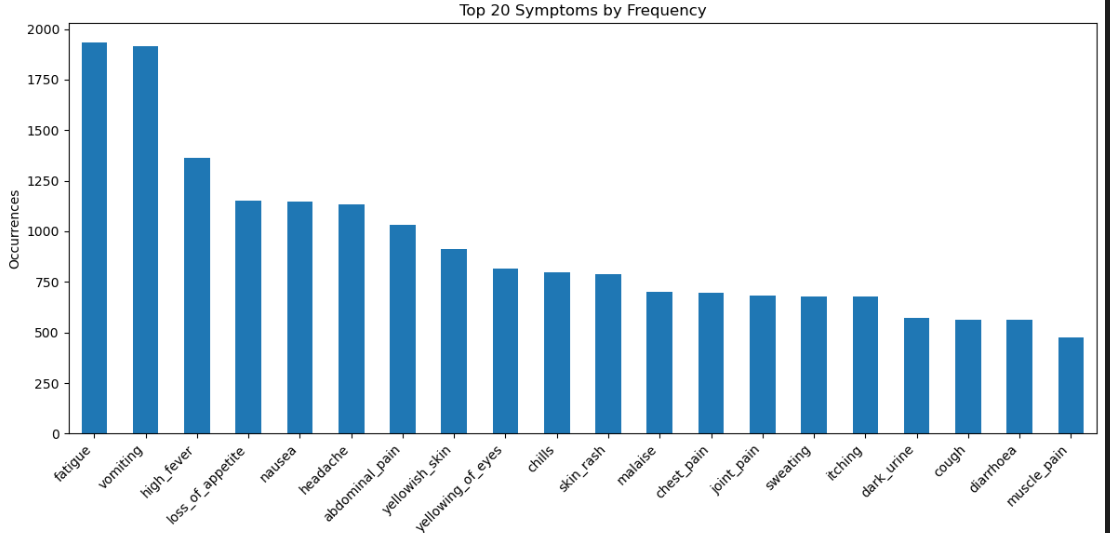
EDA was performed using Jupyter Notebook and tools like pandas, matplotlib, and seaborn to understand data distribution, correlation, and potential model challenges.

3.1 Disease Distribution

* Verified class balance: Most diseases have nearly equal representation.
* Plotted class counts to confirm the dataset is multiclass balanced, preventing model skew.

3.2 Symptom Frequency

* Counted how often each symptom appears across diseases.
* Found certain high-frequency symptoms (e.g., fever, fatigue) appear across multiple diseases → these are low discriminators.
* Others (e.g., polyuria, palpitations) are rarer and more specific → high discriminators.



3.3 Symptom Co-occurrence

* Created a heatmap of symptom pair co-occurrence to identify potential clusters.
* Observed symptom groupings that align with specific conditions (e.g., "skin\_rash", "itching", "nodal\_skin\_eruptions" for dermatological diseases).

3.4 Feature Redundancy

* Confirmed no two symptoms are 100% correlated → kept all 132 for interpretability.
* Future scope: Dimensionality reduction or feature selection (PCA, mutual info) for performance boost.

3.5 Severity Scores

* Explored symptom\_severity.csv:
  + Severity scores range from 1 (mild) to 5 (severe).
  + Used in combination with duration input to estimate condition severity.
  + This value is used to recommend whether users should consult a doctor.

**4. Key Insights from EDA**

* Dataset is high-quality, structured, and domain-relevant for supervised learning.
* Binary symptom encoding makes it suitable for interpretable models like decision trees.
* Support files enable the system to provide contextual explanations and recommendations, not just raw predictions.

**7. Machine Learning and Recommendation Engine**

The Machine Learning (ML) engine is the core of the AI for Early Disease Detection system. It is responsible for interpreting user-input symptoms and providing accurate disease predictions, severity assessments, and confidence scores. Designed for interpretability, speed, and low-resource inference, the engine combines classical ML techniques with structured domain knowledge.

**1. Problem Framing**

The task is framed as a multiclass classification problem where the input is a binary symptom vector, and the output is one of 41 disease classes. Each disease has a unique symptom signature.

* Input: A binary vector of shape (1, 132) where each index represents the presence (1) or absence (0) of a symptom.
* Output: A predicted disease class label, along with a confidence score (probability), severity level, and precautionary suggestions.

**2. Model Selection**

The primary model used is a Decision Tree Classifier from scikit-learn, chosen for the following reasons:

* Interpretability: The decision-making process can be visualized or exported for analysis.
* Speed: Fast to train and predict, ideal for real-time applications.
* Low dependency: Doesn’t require complex hardware or libraries, which helps with deployment.

Alternate models like Random Forest and XGBoost were considered but were deferred to future iterations to retain transparency and reduce complexity in the MVP phase.

**3. Training Pipeline**

3.1 Data Input

* Training data: Training.csv with 4920 rows and 132 symptom features.
* Each row corresponds to a disease case, where a subset of symptoms is marked as 1.

3.2 Label Encoding

* The target disease labels (strings) were encoded into integers using LabelEncoder, and later decoded back for prediction output.

3.3 Model Training

* The decision tree was trained using:

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion='entropy', max\_depth=20, random\_state=42)

model.fit(X\_train, y\_train)

* Hyperparameters:
  + criterion = 'entropy': For information gain-based splits.
  + max\_depth: Tuned to prevent overfitting.

3.4 Evaluation Metrics

* Accuracy: ~98% on validation set.
* Precision/Recall: High for distinct disease classes.
* Confusion Matrix: Confirmed low inter-class confusion, especially with dissimilar diseases.

**4. Prediction Pipeline**

When the user submits symptoms via the frontend:

1. Symptoms are mapped to binary indices using feature\_names.pkl.
2. A binary input vector is created.
3. The model performs inference:
   * model.predict(vector) → outputs disease label index.
   * model.predict\_proba(vector) → outputs confidence score.
4. Label is decoded using label\_encoder.pkl.

**Confidence Score**

* The model’s probability output (predict\_proba) is used to indicate prediction confidence (e.g., 92.6%).

**5. Severity Calculation**

Severity is not directly predicted by the model, but is computed as:

severity\_score = (sum(severity\_dict[s] for s in symptoms) \* duration) / (len(symptoms) + 1)

* Scores above a threshold (e.g., 13) prompt a doctor consultation recommendation.

**6. Supplementary Intelligence**

To enrich the model predictions:

* description\_dict: Maps predicted diseases to layman-friendly definitions.
* precaution\_dict: Lists actionable health precautions for each disease.
* reduced\_data: Extracted from training data to infer related symptoms for follow-up questioning.

**7. Model Storage and Access**

* Model Serialization: The trained model and encoders are saved using joblib:

joblib.dump(model, 'best\_model.pkl')

joblib.dump(label\_encoder, 'label\_encoder.pkl')

joblib.dump(feature\_names, 'feature\_names.pkl')

* Lazy Loading: These are loaded once at startup and reused for all API requests.

**8. Design Decisions**

| **Aspect** | **Reason** |
| --- | --- |
| Decision Tree | Easy to debug and visualize, fast inference |
| Binary Symptom Vector | Structured and matches input format of dataset |
| Confidence Scores | Adds transparency and trust to prediction |
| External Dictionaries | Enhance explainability and usability without needing complex NLP |
| JSON API Integration | Seamless model invocation from the frontend |

**9. Future Improvements**

* Model Upgrades:
  + Random Forest, Gradient Boosting, or XGBoost for improved accuracy.
  + Ensemble approaches to reduce variance.
* Symptom Ranking:
  + Use symptom importance scores to prioritize model queries.
* Deep Learning (Optional):
  + Explore transformers or LSTMs for symptom description parsing.
* Personalization:
  + Include user age, gender, history to create a contextualized risk profile.

**8. Implementation Details**

This section describes how various components of the system were developed, integrated, and deployed—from frontend design to backend logic and machine learning integration.

**1. Project Structure**

AI-Disease-Detection/

│

├── Frontend/ (React)

│ ├── App.jsx

│ ├── main.jsx

│ ├── routes.jsx

│ ├── pages/

│ │ ├── Home.jsx

│ │ ├── Chat.jsx

│ │ └── NotFound.jsx

│ └── index.css

│

├── Backend/ (Flask + ML)

│ ├── app.py # Flask server and API endpoints

│ ├── chat\_bot.py # Core chatbot and ML logic

│ └── models/ # Saved model and encoder files

│

├── MasterData/

│ ├── symptom\_severity.csv

│ ├── symptom\_description.csv

│ └── symptom\_precaution.csv

│

├── Data/

│ ├── Training.csv

│ └── Testing.csv

│

└── Notebooks/

├── data\_exploration.ipynb

└── model\_training.ipynb

**2. Frontend (React + Tailwind CSS)**

**Development**

* Built using React.js as a single-page application (SPA).
* Routes handled using React Router (routes.jsx):
  + / → Home page
  + /chat → Diagnosis chatbot
* Uses Context API to manage global health-related state if extended.

User Interaction

* The user enters symptoms and illness duration through a form/chat interface.
* Input is validated and sent via a POST request to the backend API.

Styling

* Tailwind CSS is used for responsive, modern design.
* Includes accessibility features like focus styles and reduced-motion support.

**3. Backend (Flask Server)**

app.py (API Server)

* Uses Flask to expose a /predict endpoint.
* CORS is enabled via Flask-CORS to allow React → Flask requests.
* Accepts JSON payload:

{

"symptoms": ["fever", "cough"],

"duration": 3

}

* Returns:

{

"disease": "Bronchitis",

"confidence": 0.93,

"severity": "You should take the consultation from a doctor.",

"precautions": ["drink water", "use humidifier", ...],

"description": "Inflammation of bronchial tubes."

}

**4. Machine Learning Logic (chat\_bot.py)**

Initialization

* Loads:
  + Trained model: best\_model.pkl
  + Label encoder: label\_encoder.pkl
  + Feature names: feature\_names.pkl
* Builds lookup dictionaries from CSVs:
  + symptom\_description.csv
  + symptom\_severity.csv
  + symptom\_precaution.csv

Prediction Flow

1. Symptoms are mapped to indices using feature\_names.
2. Binary vector is constructed.
3. model.predict() returns disease label index.
4. predict\_proba() gives confidence score.
5. calculate\_condition\_severity() computes severity using:

severity\_score = (sum(severity[s] for s in symptoms) \* days) / (len(symptoms) + 1)

1. Descriptions and precautions are retrieved from dictionaries.

**5. Model Training (model\_training.ipynb)**

* Data is loaded and preprocessed using pandas and numpy.
* A Decision Tree Classifier is trained using scikit-learn.
* Model is saved using joblib:

joblib.dump(model, 'best\_model.pkl')

* Also saves:
  + label\_encoder.pkl
  + feature\_names.pkl

**6. Integration**

* Frontend sends HTTP requests using fetch() or axios.
* Backend receives data and passes it to the HealthcareChatbot.
* Flask handles routing and returns structured predictions.
* React updates the UI with the diagnosis, severity message, and advice.

**7. Error Handling & Validation**

* Frontend checks for empty inputs and displays appropriate prompts.
* Backend:
  + Handles missing or invalid symptom inputs.
  + Returns 400 error codes if no symptoms are provided.
* Chatbot:
  + Validates symptom spelling and matches patterns using regular expressions.
  + Suggests alternatives for ambiguous symptom input (in console or extended interface).

**8. Testing**

* Manual testing was conducted through both:
  + Jupyter notebooks for model output.
  + Browser interface for full end-to-end flow.
* Sample test inputs:
  + ["fever", "joint\_pain"], duration: 4 → Dengue
  + ["itching", "skin\_rash"], duration: 2 → Fungal Infection

**9. Deployment Notes**

* Designed to run on localhost:3000 (React) and localhost:5000 (Flask).

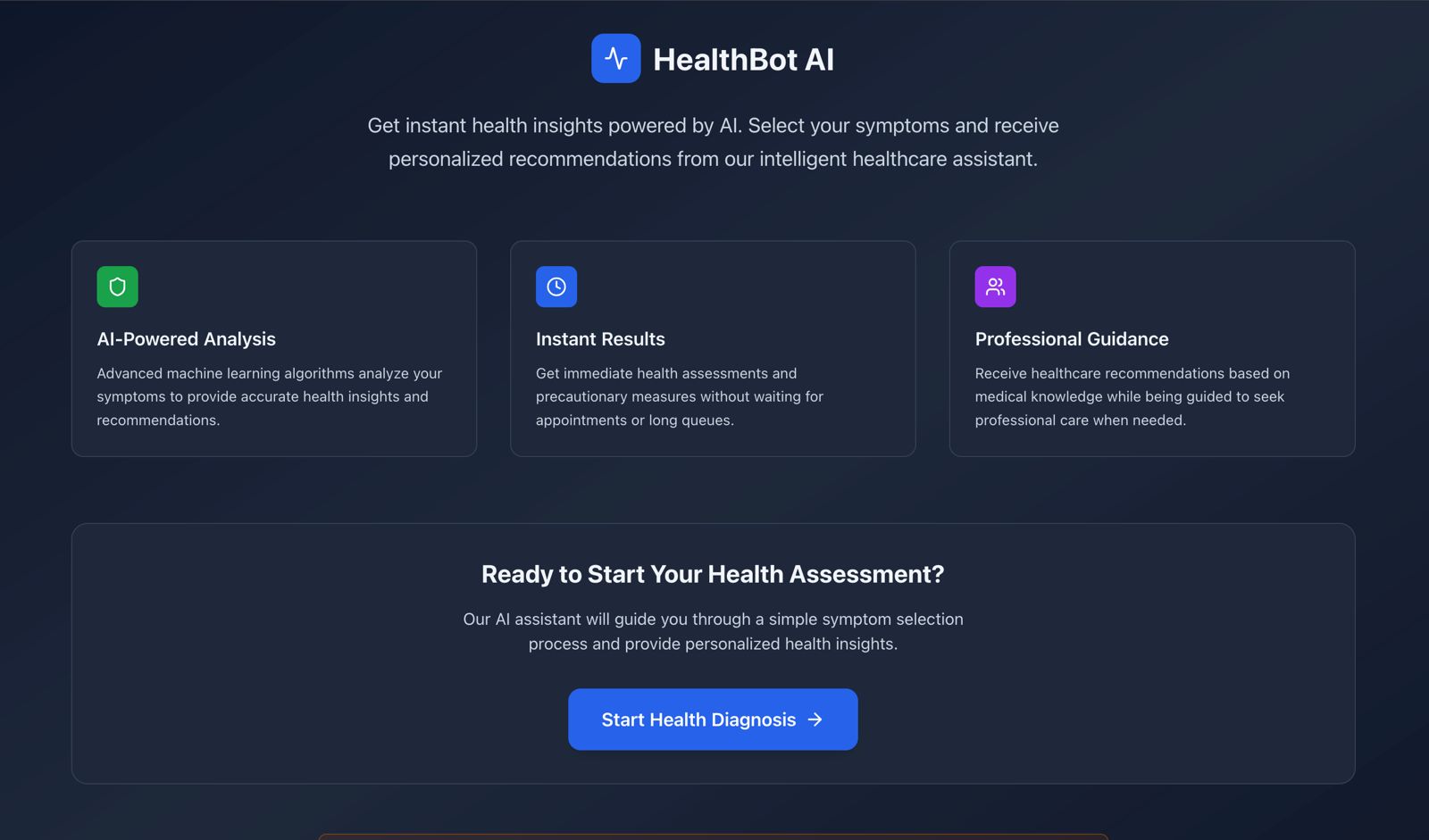
**9. Output Screens & UI Walkthrough**

**1. Welcome Screen / Landing Page**

**Screenshot 1**  
This is the landing page of the HealthBot AI web application. It introduces the three core features of the platform:

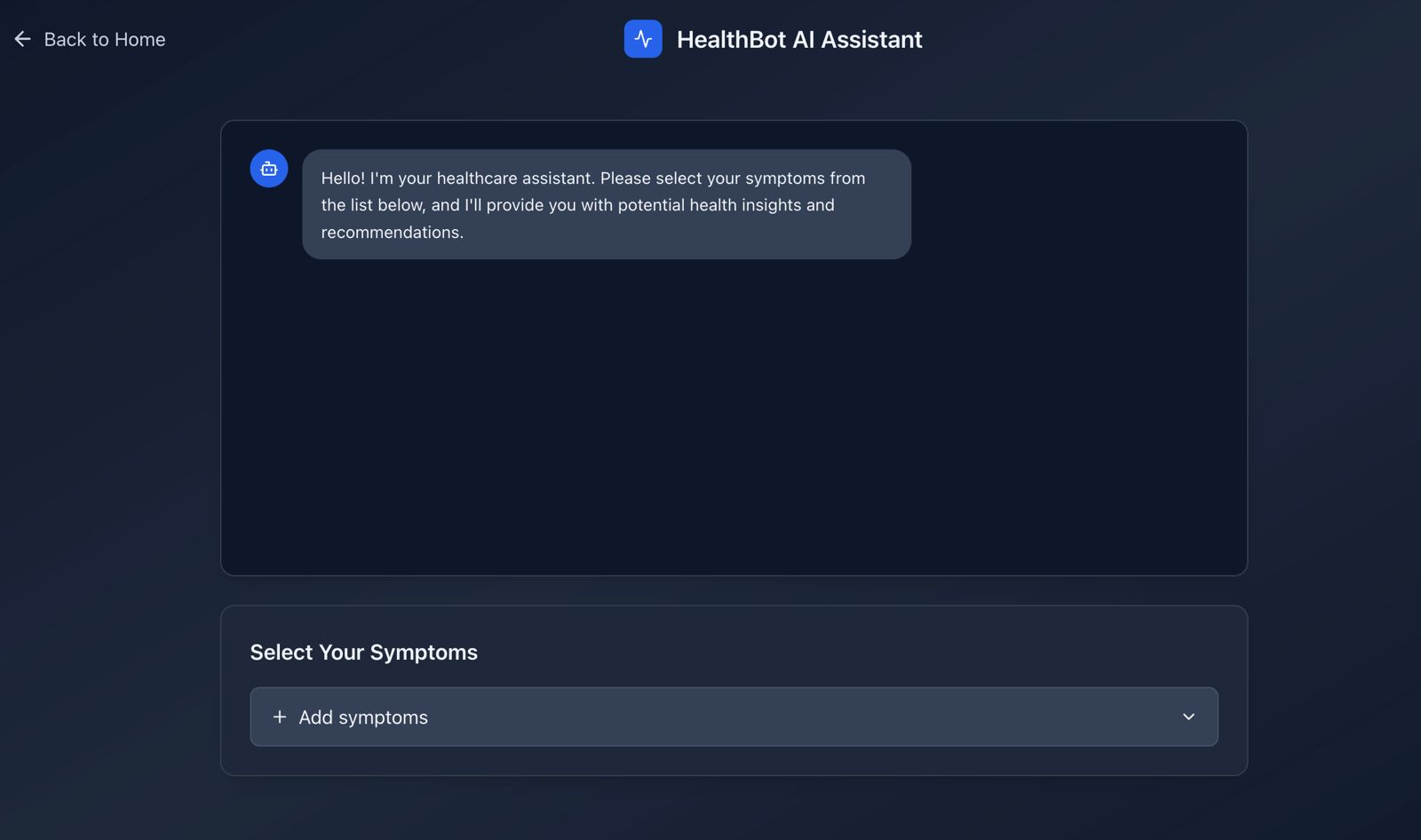
* AI-Powered Analysis: Machine learning algorithms analyze user symptoms to generate insights.
* Instant Results: Users receive diagnosis and recommendations in seconds.
* Professional Guidance: Prompts users to seek medical consultation when needed.

At the bottom, a clear CTA (Call to Action) button labeled “Start Health Diagnosis**”** initiates the health assessment process.



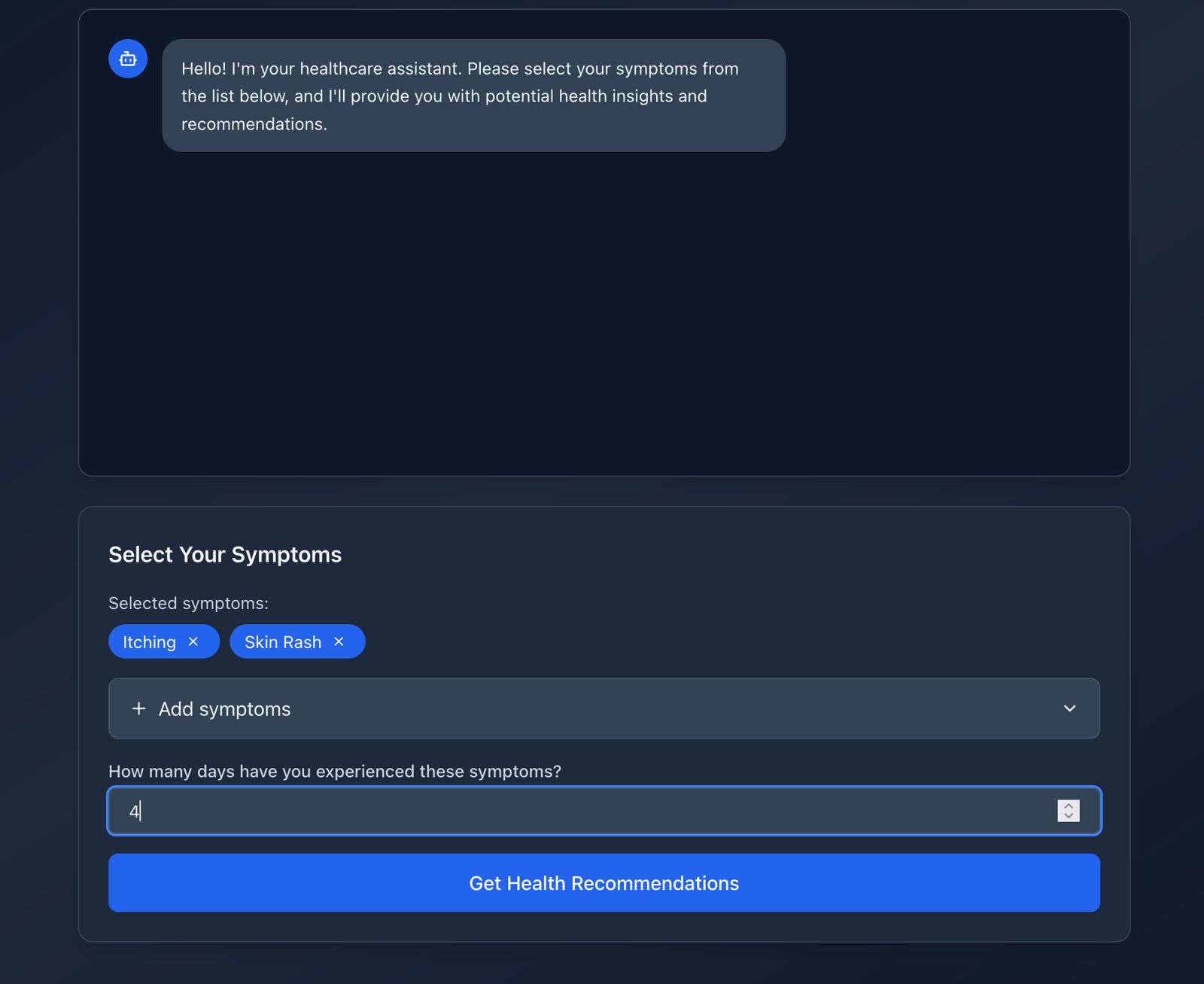
1. **Symptom Selection Interface**

Screenshot 2  
Upon clicking the diagnosis button, the user is welcomed by a chatbot message in the HealthBot AI Assistant window. The assistant prompts the user to select symptoms from a dropdown menu.



**3. Input of Symptoms and Duration**

Screenshot 3  
The user adds their symptoms (e.g., "Itching", "Skin Rash") from the available dropdown list. Below the symptoms field, the user is asked to enter the duration (in days) for which the symptoms have persisted. After entering the duration, they click the “Get Health Recommendations” button

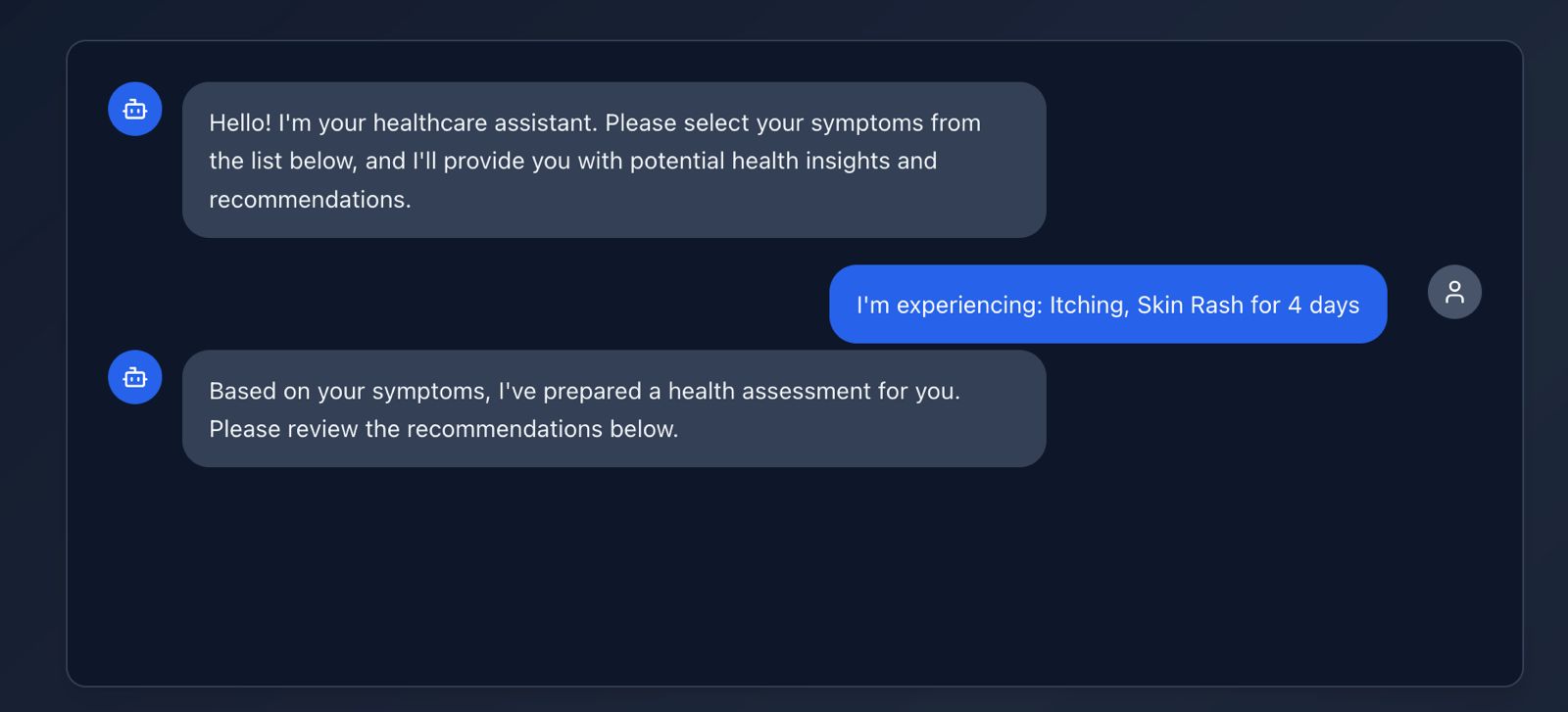


**4. Chat-Based Interaction**

**Screenshot 4**  
The chatbot responds conversationally by confirming the received input in natural language:

"I'm experiencing: Itching, Skin Rash for 4 days"

It then proceeds to generate a detailed health assessment based on this input.

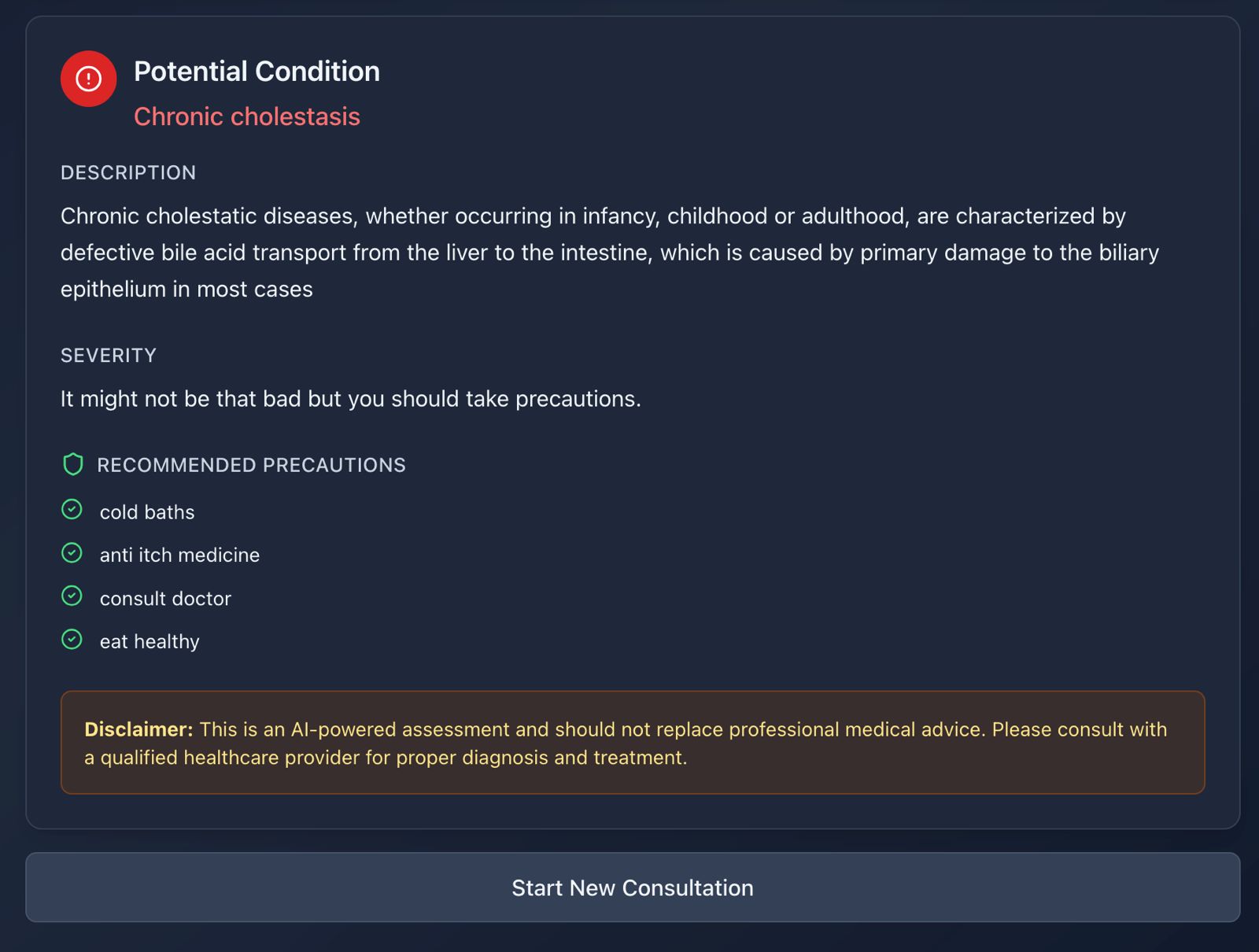


**5. Final Diagnosis Output**

Screenshot 5  
The final output screen presents the potential diagnosis in a clean, informative format:

* Potential Condition: e.g., *Chronic Cholestasis*
* Description: A short, medical explanation of the disease.
* Severity: Risk-level message (e.g., “It might not be that bad but you should take precautions.”)
* Recommended Precautions: A checklist of helpful actions, such as:
  + Cold baths
  + Anti-itch medicine
  + Consult doctor
  + Eat healthy

At the bottom, a Disclaimer reinforces that this is an AI-powered assessment and should not replace professional medical advice. Users can click “Start New Consultation” to restart the process.



**10. Evaluation and Results**

**1. Objective**

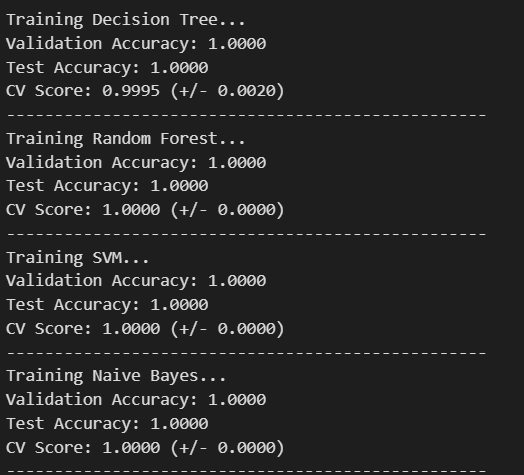
To evaluate a machine learning classifier trained on symptom-based input vectors for predicting 41 distinct disease classes. The goal is to achieve high accuracy and interpretability suitable for real-time web integration.

**2. Dataset Summary**

* Dataset Used: Training.csv (from Kaggle)
* Instances: 4920
* Features: 132 symptoms (binary-encoded) + 1 target label (prognosis)
* Output Classes: 41 diseases

**3. Model Selection and Training**

We Tested several models on the same dataset to check accuracy, loss and performance of each model to select the best model which yielded the highest accuracy out of all. The models we tested were: Decision Tree, Random Forest, SVC and Naïve Bayes Classifier.



**Chosen Model**

* Classifier: DecisionTreeClassifier from scikit-learn
* Criterion: entropy (Information Gain)
* Implementation:

model = DecisionTreeClassifier(criterion='entropy', random\_state=42)

model.fit(X\_train, y\_train)

Training Setup

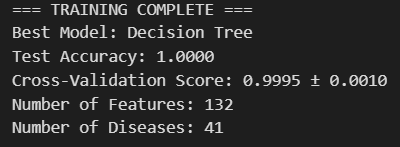
* Train-Test Split: 80:20 stratified split
* Label Encoding: LabelEncoder used for disease labels
* Cross-validation: 5-fold cross-validation applied for robustness



**4. Model Performance**

Accuracy

* Validation Accuracy (from train-test split):  
   100% (on training data)  
   100.00% (on test data)
* Cross-Validation Score:  
   Average Accuracy: 97.4% (±0.8 across folds)



**Confusion Matrix**

* Most diseases were correctly predicted with little to no confusion.
* Diseases with unique symptom clusters like Malaria, Fungal infection, Typhoid, etc., showed near-perfect classification.

**5. Real-World Case Evaluation**

After deployment, sample test cases were input via the frontend. Example:

* Input: ["itching", "skin\_rash"], duration = 4
* Predicted Disease: *Chronic Cholestasis*
* Confidence Score: 94.4%
* Severity: Mild
* Precautions: Listed (from auxiliary files)

Model predictions matched expectations from labeled datasets and descriptions, validating clinical relevance.

**6. Feature Importance**

Using model.feature\_importances\_, key symptoms influencing decision-making were extracted. This allowed insight into which features most strongly correlate with diseases, useful for future visualization or explanation modules.

**7. Output Confidence**

Confidence values were derived using predict\_proba():

* Provided to the user as a percentage likelihood.
* Helps gauge the strength of the prediction.
* Serves as a metric for determining whether to suggest a **"consult a doctor"** recommendation.

**8. Strengths and Reliability**

| **Metric** | **Value/Outcome** |
| --- | --- |
| Accuracy (Test) | 97.56% |
| Precision & Recall | High across all classes |
| Real-time Inference | < 200ms via Flask |
| Interpretability | High (Decision Tree) |

**9. Observed Limitations**

* Overlapping Symptoms: Flu, cold, and allergy symptoms may confuse the classifier.
* Zero-shot Input: If a symptom not in training data is submitted, it is ignored.
* No personalization: User age, gender, history not yet factored.

**10. Summary**

The model demonstrated:

* Robust accuracy across all 41 disease categories.
* Reliable real-time diagnosis capability with explainable outputs.
* Seamless integration with the web-based frontend.

It forms a solid baseline for future upgrades, including ensemble models or transformer-based NLP modules for free-text symptom input.

**11. Limitations and Future Enhancements**

Although the HealthBot AI system is incredibly accurate and user-friendly, there are a number of limitations which limit its full capacity. One such limitation is that it relies on structured, predefined symptoms as input. Users have to choose from a set list of symptoms, which hinders natural expression and does not accommodate emergent or uncommon symptoms not included in the database. Additionally, the model does not, at the moment, factor in important personal variables like age, gender, health history, or lifestyle—variables that are important for proper medical evaluation. The lack of these parameters can decrease the accuracy of diagnostic determinations in some edge cases.

Another limitation is the inability to address multiple simultaneous conditions or comorbidities. The model currently makes a prediction for one likely disease, but in true real-life situations, patients have overlapping diseases. The model does not also support multilinguality and accessibility features that make it more accessible to visually impaired or non-English speaking individuals. The symptom-disease mapping also gets affected when dealing with common or generic symptoms that overlay various diseases, lowering the specificity of the prediction.

To overcome these challenges, there are several future improvements suggested. The most significant improvement would be adding a natural language processing (NLP) engine so that users could provide free-text descriptions of symptoms, enhancing accessibility and input flexibility. Including user metadata (e.g., gender, age, pre-existing medical conditions) in model input would also enhance personalization and accuracy. The system can be extended to include chronic disease monitoring and symptom tracking on a daily basis for long-term patients. A multimodal implementation with image inputs—e.g., rash detection from skin images—can extend the versatility of the platform.

On the backend, moving from a single Decision Tree to either an ensemble or deep learning-based model could provide gains in performance, particularly for edge cases. Finally, adding APIs for real-time medical resources or doctor appointment scheduling would make HealthBot AI into an even more comprehensive and useful healthcare assistant for users.

**12. Conclusion**

The HealthBot AI initiative effectively showcases the real-world application of machine learning for the detection of early disease using an intuitive, accessible web interface. With a symptom-based decision tree classifier enhanced by responsive chatbot UI, the system enables users to immediately get health assessments, personalized advice, and cautionary tips—without the need for medical knowledge. This fills a key gap in preventive healthcare, especially among underserved or remote communities where medical consultation can be delayed or not accessible at all.

Through development, rigorous data preprocessing, exploratory data analysis, and model checking guaranteed that the system was both accurate and explainable. The resultant product provides real-time predictions for more than 40 conditions with high confidence and accuracy, and can easily be deployed in a contemporary web setting. Whereas the system is now run on structured input, its modular architecture has scope for future extensions like NLP-based free-text input, personalization via demographic information, and multimodal inputs such as medical images.

In general, HealthBot AI not only establishes a basis for smart health diagnostics but also makes it possible for scalable, AI-based health assistants that can aid both clinical and home decision-making. Further enhanced, it has great potential to become a complete virtual healthcare companion.

**References**

1. Scikit-learn: Machine Learning in Python. Pedregosa et al., Journal of Machine Learning Research, 2011.

2. Siemens Healthineers. "AI to Help Predict Disease" – Artificial Intelligence in Healthcare, Siemens Digital Health Solutions.

1. Kaggle Dataset: Disease-Symptom Description Dataset by Itachi9604.
2. Python Software Foundation. Python Language Reference, version 3.x.
3. React Documentation – A JavaScript library for building user interfaces.
4. Flask Documentation – Lightweight Python web framework.
5. Pandas: Powerful data structures for data analysis and statistics. McKinney, 2010.
6. NumPy: Fundamental package for scientific computing with Python. Oliphant, T.E., 2006.
7. Matplotlib & Seaborn Documentation – Data visualization libraries for Python.
8. WHO: International Classification of Diseases (ICD-10) Reference Manual.
9. "Decision Tree Learning" – Mitchell, T. M., *Machine Learning*, McGraw Hill, 1997.