DISEASE PREDICTION WEB APPLICATION USING MACHINE LEARNING

FOO JIA YU

UNIVERSITI TUNKU ABDUL RAHMAN

DISEASE PREDICTION WEB APPLICATION USING MACHINE LEARNING

FOO JIA YU

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Software Engineering (Honours)

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

September 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Name	:	FOO JIA YU
ID No.	:	2105105
Date	:	01/07/2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Dr. Fatimah Audah binti Md. Zaki as my research supervisor and Mr Kelwin Tan Seen Tiong as my research moderator for their invaluable advice, guidance, enormous patience and constructive feedback throughout the duration of the research.

I sincerely thank my academic advisor, Dr. Khor Kok Chin, for his exceptional guidance and support throughout my degree program. I would also like to thank all the lecturers and the faculty and the departmental members from Lee Kong Chian Faculty of Engineering and Science for creating a pleasant working environment throughout my years in UTAR.

Special thanks go to my fellow classmates and friends for their collaboration, discussions, and encouragement during challenging times. Their support helped me kept motivation and progress steadily.

Finally, I am deeply grateful to my family for their unconditional love, patience, and unwavering trust. Their encouragement has been my greatest source of strength and inspiration throughout my academic journey.

ABSTRACT

In recent years, the prevalence of diseases has increased and the demand for quick diagnosis tools is growing. This has highlighted the need for machine learning-based web applications for disease predictions is important in the healthcare system for early diagnosis. This project presents the design and development of a web-based disease prediction application that employs machine learning and natural language processing technologies to assist users in identifying potential health conditions. The motivation for this project is to improving access to early diagnosis, reduce the burden on medical staff and getting general medical advice anywhere and anytime. The methodology involved develop and train machine learning models on Symptom-Disease Prediction Dataset (SDPD) to achieve precise predictions, integrate the model into web application built on Flask and React, and employ Google Gemini to generate general medical recommendations and extract symptoms. System testing was conducted through multiple testing methods, including unit testing, integration testing, user acceptance testing (UAT) and user interface design feedback collected through Google Forms. The results indicate that the machine learning model achieved a prediction accuracy at approximately 97%. User acceptance testing validated that over 90% of users rated the usability and ease of use of the system at 4 or higher on a 5-point Likert scale. The study concluded that the system successfully achieved its objectives, delivering a practical, user-friendly, and intelligent healthcare support system. However, it also acknowledged limitations such as dependence on dataset quality, lack of coverage for rare or new diseases, and multilingual support. Future work will focus on expanding dataset variety, integrating multilingual support, and incorporating of contextual health data to further enhance prediction accuracy and precision.

Keywords: Disease prediction, Machine Learning, Web Application, Large Language model, Natural Language Processing

Subject Area: QA76 – Computer Science

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
RT	Random Forest
DT	Decision Tree
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
NB	Naïve Bayes
SDLC	Software development lifecycle
API	Application programming interface
WBS	Work Breakdown Structure
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
WHO	World Health Organization

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

The World Health Organization (WHO) states that the chronic diseases such as diabetes and cardiovascular account for 74% of global deaths annually, but many chronic diseases can be prevented through early detection (WHO, 2024). Nowadays, people are more concerned about health after the breakdown of a pandemic. Rapid advances in technology and artificial intelligence (AI) are having a significant impact on the healthcare industry. Traditional diagnosis processes in the healthcare domain are time-consuming and costly. As there are many different diseases worldwide, the healthcare staff may have some limitations in their skills and knowledge of certain diseases. This may limit their ability to make early diagnoses of some particular diseases. However, machine learning-based healthcare systems may not have these limitations and can become a powerful tool for early diagnosis.

Machine learning (ML) is a subfield of Artificial Intelligence (AI) that focuses on allowing computers and machines to mimic human learning, carry out activities autonomously, and enhance the performance and accuracy of predictions through past and large datasets (IBM, 2021). Machine learning is used to train machines to handle complex data more efficiently and effectively and provide accurate results. There are two phases of the machine learning algorithm, which are Training and Testing (Sharmila *et al.*, 2024). Healthcare issues can be effectively addressed using Machine Learning Technologies. Through the use of Machine Learning, many time-consuming tasks can now be completed quickly and with minimal effort and also helps to reduce human errors.

Different diseases may have some similar or different symptoms. These overlapping symptoms may confuse medical professionals in the diagnosis process. Therefore, the disease prediction system can use machine learning to predict diseases based on patient symptoms. This can assist medical staff in making a diagnosis and reduce human errors. There were few existing applications that can predict disease based on a patient's symptoms and other features. Unlike WebMD, this system focuses on provide medical advice based on the potential diseases. This project focuses on developing a disease prediction web application by using machine learning. By using symptoms inputted or selected by users, the system can predict potential illnesses and provide recommendations for the users. This application can enhance early detection and help medical professionals and individuals in making decisions about their health.

1.2 Importance of the Study

In recent years, the prevalence of diseases has increased and the demand for quick diagnosis tools is growing. This has highlighted the need for machine learning-based web applications for disease predictions is important in the healthcare system for early diagnosis. This study focuses on developing a disease prediction web application by using machine learning. The web application analyses the user-input symptoms and diagnoses the potential disease for the user. The users can seek medical advice and recommendations earlier by analysing the symptoms early.

In addition, the disease prediction system can automatically diagnose the disease and reduce the workload of medical staff. This saves the time of healthcare professionals in diagnosing the disease one by one, therefore, they have more time to concentrate on some complex cases. This can make the diagnostic process more efficient and simpler. Besides that, it can assist healthcare professionals to make the diagnosis and reduce human errors. This is due to the reason that the symptoms of many diseases may overlap, which can confuse the medical professionals and prevent them from making a correct judgment based on the symptoms.

Furthermore, healthcare personnel may not have skills in particular areas, which will make it difficult for them to make decisions about the diseases. Therefore, the disease prediction system can assist both healthcare staff and individuals in making well-informed choices regarding their health. This study focuses on developing a web application for predicting diseases by utilizing machine learning techniques. By using the symptoms entered by users, the system can predict potential diseases and provide some recommendations to the users. This application aims at early diagnosis of diseases, reduces diagnostic errors, and can help users to make a correct judgment about illnesses. Moreover, the system can be used as a preliminary diagnosis tool to guide the users to seek appropriate medical advice and medical attention.

1.3 Problem Statement

In the past few years, disease diagnosis has frequently been dependent on the experience and knowledge of medical experts, which may occasionally result in delays or incorrect diagnoses. Moreover, some individuals who live in distant areas or have limited access to healthcare facilities may find it difficult to get medical services. Therefore, the disease prediction system can assist in early diagnosis based on symptoms entered by users. There were some problems in the current healthcare system, including:

1.3.1 Long waiting times for patients to get diagnosis

One of the biggest problems faced by current healthcare systems is the long waiting times for patients to receive a diagnosis. According to Datuk Dr N. Marimuthu, the waiting time for patients in public hospitals is up to 3 hours and should be reduced to 30 minutes as in the public health clinics (BERNAME, 2024). This is because of the large number of patients and limited availability of medical staff. As a result, some patients may choose to self-medicate rather than wait for a professional diagnosis, delaying proper treatment since the treatment process often involves long waiting times. Besides that, these delays may also lead to worsening conditions for patients, particularly those with chronic illnesses. According to Newman-Toker et al. (2024), the diagnostic errors including misdiagnosis and delayed diagnosis resulted in 795,000 serious harms. These included 371,000 deaths and 424,000 permanent disabilities, underscoring the severity of the human toll. The machine learning-based disease prediction system can prevent this issue by providing users with an initial assessment of a user before visiting a healthcare facility and providing assistance to medical staff to improve diagnostic accuracy.

1.3.2 Increased Risk of Human Errors

Furthermore, human error is inevitable in the healthcare system. Although healthcare professionals have specialized knowledge and experience, they may also make some mistakes when diagnosing the disease because the symptoms of the patients may overlap with those of other diseases. This may result in misdiagnosis, leading to incorrect treatments of the patients and potentially causing severe complications. About 200,000 patients die each year from preventable medical errors (Kavanagh et al., 2017). Moreover, some of incorrect patients' records may also kept in the healthcare system due to human mistakes. This may mislead the doctors when diagnosing the disease for patients. Utilizing machine learning models can minimize the likelihood of errors and increase the accuracy of the disease predictions by using large datasets of medical records.

1.3.3 Limited accessibility for remote areas

Due to the inadequate numbers of hospitals and doctors in some remote or rural areas, people living there have limited access to healthcare services depending on their region. If they want to have access to comprehensive and refined healthcare services and facilities, they may have to travel to other regions or countries, which is time-consuming and costly. Rural residents often face barriers to health care that limit their ability to access needed medical services (Rural Health Information Hub, 2024). If the patients suffer from serious diseases, they may not be able to get timely diagnosis in rural areas. The disease prediction system can help to prediction their disease early based on the symptoms and can give some medical advice for them. This can lower the chance of serious complications and provide access to the basic healthcare information. In addition, the system can guide individuals to seek medical attention when they may be suffering from some disease.

1.4 Aim and Objectives

The primary aim of this project is to develop a web-based application that can allow users to predict their disease based on user-inputted or selected symptoms. This system is based on machine learning and the model are trained with the selected dataset to ensure the accuracy of the results. The selected dataset is Symptom-Disease Prediction Dataset (SDPD), which is sourced from Mendeley. Besides that, the disease prediction system will serve as a tool for early diagnosis of disease, helping users to know the potential disease they may be suffering from based on their symptoms when they feel unwell. Moreover, the system will provide users with possible solutions for further action, better understanding, or medical treatment after a diagnosis of a disease they may have. The disease prediction project is designed to accomplish the following objectives:

1. To develop and train a machine learning model capable of predicting specific diseases, achieving a prediction accuracy of 85% or higher on the test dataset.
2. To design a user-friendly web application and evaluate its usability through User Acceptance Testing (UAT), ensuring that at least 90% of users rate its ease of use as 4 or higher on a 5-point Likert scale.
3. To design and test different prompts for large language model (Google Gemini), evaluating their effectiveness in advice generation and validate the outputs against trusted medical sources.

1.5 Scope and Limitation of the Study

This project aims to develop a web application to predict potential diseases for users, which also provides medical advice and recommendations on the potential diseases users may have and allows users to input their symptoms for prediction. The scope of this project is including:

i. **User-friendly web interface**

   A simple and responsive web interface where users can enter their symptoms for prediction. In addition, provides two options for users to enter the symptoms which are a predefined dropdown list of symptoms or manually enter symptoms via a free-text field.

ii. Machine learning model training

A trained machine learning model is able to analyse the user input or selected input symptoms and predict the potential diseases users may have. The model utilizes Random Forests algorithm for the prediction process.

iii. Server-side applications

A server-side application that can process user inputs, interacts with the machine learning models and returns the predictions to the user. Implement APIs to handle the data flow between the front end and the machine learning model.

iv. Database to store user information

Develop a database for storing the user history such as user inputs and predicted results. This can help user to track their symptoms and potential diseases over time.

v. Included Diseases:

There are 41 diseases included in the system, as listed in Table 1.1 and Table 1.2 presents the 131 symptoms supported by the system.

Table 1.1: List of diseases included in the system

Fungal Infection	Allergy	GERD	Chronic Cholestasis	Drug Reaction
AIDS	Diabetes	Gastroenteritis	Bronchial Asthma	Hypertension
Migraine	Peptic Ulcer Disease	Cervical Spondylosis	Paralysis (brain hemorrhage)	Jaundice
Malaria	Chickenpox	Dengue	Typhoid	Hepatitis A
Hepatitis B	Hepatitis C	Hepatitis D	Hepatitis E	Alcoholic Hepatitis

Tuberculosis	Common Cold	Pneumonia	Dimorphic Hemmorhoids (piles)	Heart Attack
Varicose Veins	Hypothyroidism	Hyperthyroidism	Hypoglycemia	Osteoarthritis
Arthritis	Vertigo	Acne	Urinary Tract Infection	Psoriasis
Impetigo				

Table 1.2: List of the 132 symptoms supported by the system.

Itching	Skin rash	Nodal skin eruptions	Continuous sneezing	Shivering	Chills
Joint pain	Stomach pain	Acidity	Ulcers on tongue	Muscle wasting	Vomiting
Burning micturition	Spotting urination	Fatigue	Weight gain	Anxiety	Cold hands and feets
Mood swings	Weight loss	Restlessness	Lethargy	Patches in throat	Irregular sugar level
Cough	High fever	Sunken eyes	Breathlessness	Sweating	Dehydration
Indigestion	Headache	Yellowish skin	Dark urine	Nausea	Loss of appetite
Pain behind the eyes	Back pain	Constipation	Abdominal pain	Diarrhoea	Mild fever

Yellow urine	Yellowing of eyes	Acute liver failure	Fluid overload	Swelling of stomach	Swelled lymph nodes
Malaise	Blurred and distorted vision	phlegm	Throat irritation	Redness of eyes	Sinus pressure
Runny nose	Congestion	Chest pain	Weakness in limbs	Fast heart rate	Pain during bowel movements
Pain in anal region	Bloody stool	Irritation in anus	Neck pain	Dizziness	Cramps
Bruising	Obesity	Swollen legs	Swollen blood vessels	Puffy face and eyes	Enlarged thyroid
Brittle nails	Swollen extremeties	Excessive hunger	Extra marital contacts	Drying and tingling lips	Slurred speech
Knee pain	Hip joint pain	Muscle weakness	Stiff neck	Swelling joints	Movement stiffness
Spinning movements	Loss of balance	Unsteadiness	Weakness of one body side	Loss of smell	Bladder discomfort

Foul smell of urine	Continuous feel of urine	Passage of gases	Internal itching	Toxic look (typhos)	Depression
Irritability	Muscle pain	Altered sensorium	Red spots over body	Belly pain	Abnormal menstruation
Dischromic patches	Watering from eyes	Increased appetite	Polyuria	Family history	Mucoid sputum
Rusty sputum	Lack of concentration	Visual disturbances	Receiving blood transfusion	Receiving unsterile injections	Coma
Stomach bleeding	Distention of abdomen	History of alcohol consumption	Blood in sputum	Prominent veins on calf	Palpitations
Painful walking	Pus filled pimples	Blackheads	Scurring	Skin peeling	Silver like dusting
Small dents in nails	Inflammatory nails	Blister	Red sore around nose	Yellow crust ooze	

There also some limitations in this project, including:

1.5.1 Limitation

i. **Not replace the medical professional**

This system is intended only as an assistant for medical professionals or individuals in the early diagnosis of the potential disease and does not replace the medical professional for given medical treatment. The system is only used for preliminary evaluation.

ii. **Not cover rare or new disease**

The system does not cover all the diseases, especially is some rare or new diseases. The symptoms and conditions of the rare or new disease may not be included in the training dataset, so it may be unable to detect the disease based on the symptoms.

iii. **Accuracy depends on the quality and quantity of dataset**

The reliability of the disease prediction model depends on the dataset, therefore the accuracy of the prediction results depends on the correctness and size of the dataset. If the dataset is limited and incorrect, the accuracy of predicted results will be low and inefficient.

iv. **External factors**

There may be some unpredictable factors to be considered when use the disease prediction system such as the breakdown of pandemics. The pandemic outbreaks may not have enough datasets to train the model to predict new pandemics disease in a short period of time.

v. **Only supports English input and output**

The system only accepts and generates the prediction results in English. This may limit the use of the system for non-English users and reduce the usability of the system in multilingual environment. The non-English users may require external translation tools to use the system, which may affect the accuracy and user experience.

1.6 Proposed Solution

The aim of this project is to develop a disease prediction web application using machine learning based on user-inputted symptoms to address the problems outlined in the problem statement. The features of the web application include entering symptoms, accessing potential disease outcomes, medical advice, personal profiles and updating personal profiles. The system enables the user to input their symptoms through two different methods. The first is to select from a predefined list of 132 symptoms and the second is to enter symptoms in free-text format and extracted the symptoms using large language model. The user is free to choose the symptom input option.

A Random Forest model is being trained on selected datasets to predict potential outcomes. In addition, the user can also track their symptoms over time by viewing the historical data. By using Large Language Model, the system also provides actionable information such as self-care tips or general medical advice for different diseases, enabling users can take appropriate action before seeking medical attention. By providing the initial assessment, the web application aims to reduce waiting times and human errors, assist medical staff in prioritizing cases and provide early diagnosis for patients in rural areas with high rates of diagnostic delay. The Figure 1.1 shows the system design overview.

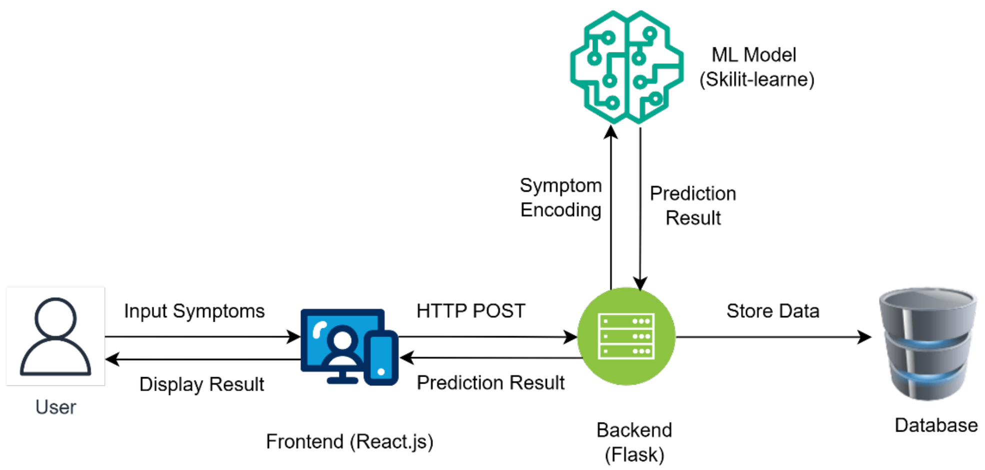


Figure 1.1: System Design Overview

1.7 Project Approach

This project is carried out in several structured phases to ensure systematic development process. The Software Development Life Cycle used in this project is Waterfall Methodology. There are 6 phases in the Waterfall Methodology. The first phase is project requirements analysis. This phase involves gathering project requirements by analysing existing similar disease prediction web applications. The second phase is system design. This phase involves planning the overall system structure and outlining the data flow of the project. In addition, the development of machine learning models includes dataset selection, data preprocessing, training the model, evaluating the model, and then integrating it with the web application. The fourth phase involves designing and developing a responsive web application using React for the frontend and Flask for the backend. The trained machine learning model will be integrated with the web application to process user inputs and return predicted results. In the subsequent phase, the system will be tested including functionality testing, usability testing, user acceptance testing and so on. Figure 1.2 shows the Waterfall Methodology used for this project. The Waterfall is suitable for progress tracking and deliverables. This helps in estimating the project timeline and budget well.

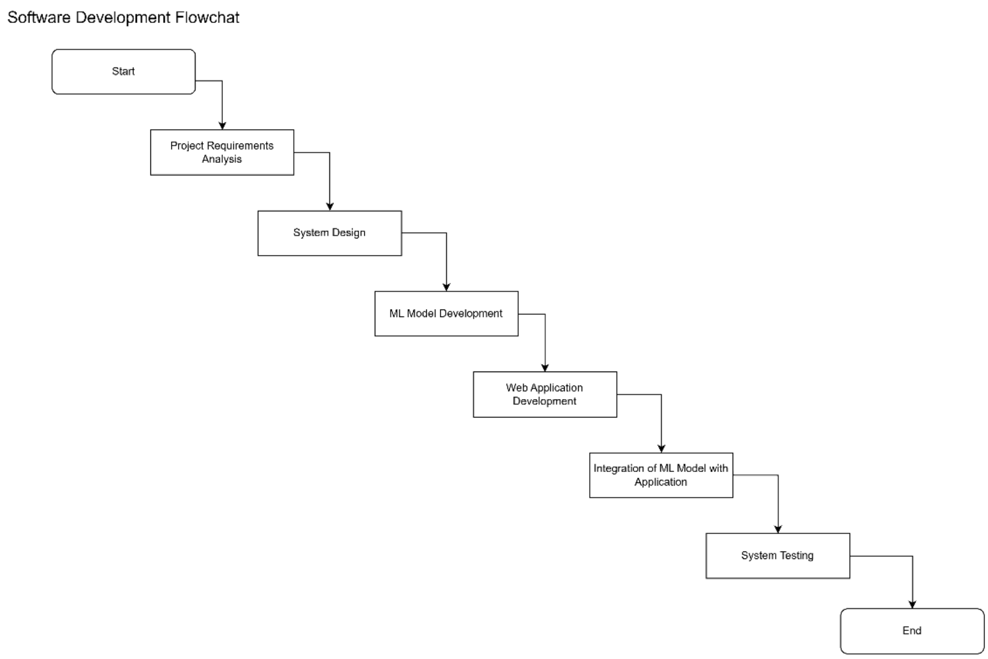


Figure 1.2: Waterfall Methodology

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The demand for disease prediction application in healthcare field is increasing, and the disease prediction web application useful for early diagnose of the potential diseases. By utilizing the disease prediction system, can reduce the reliance on the expert analysis and provide advance method to identifying the disease based on the symptoms. This chapter presented a depth review of related work to disease prediction using machine learning, comparisons of different models, comparisons of existing web applications of disease prediction, and get suitable evaluation metrics for assessing predictive models.

2.2 Research and Comparison Model

Machine learning has been utilized extensively for early diagnosis in healthcare. This part focuses on some researching and comparing the different models.

2.2.1 Research Models

Sangeetha *et al*. (2024) proposed a screening system to identify the disease based on user-inputted symptoms by using 3 models which are Decision Tree, Random Forest and Naive Bayes. Among these models, the decision tree has the highest accuracy, which is 97.53%, followed by Random Forest at 95.66% and Naïve Bayes is 93.75%. Based on the performance, the decision tree algorithm was chosen to integrate with the user interface in their system.

In addition, Ansarullah *et al*. (2022) proposed a risk prediction model that can initially detect the heart disease by using multiple machine learning techniques. The features used in this study contain 12 attributes entered by the user for early prediction of heart disease, such as demographical inputs and behaviour inputs. There are 5 models used by authors which are K-Nearest Neighbor, Naïve Bayes, Decision Tree, Random Forest, and Support Vector Machine. In this paper, the risk models were evaluated by different measures. Based on various performance metrics, the Random Forest model has the best performance among these models with an accuracy of 84%, a sensitivity of 85%, a specificity of 83%, an error rate of only 13%, and a precision of 85%.

Furthermore, George *et al*. (2024) proposed a system that utilizes machine learning to provide disease predictions, complemented by chatbot, and doctor-patient appointment system. In this paper, the authors predicting 4 diseases which are heart disease, kidney disease, brain tumour and breast cancer. The features used are age, blood pressure, cholesterol level, image of tumour and mammographic image analysis. The algorithms used in the prediction are Support Vector Machine (SVM), Random Forest, K-Nearest Neighbor (KNN) and Convolutional Neural Networks (CNN). For kidney disease and heart disease diagnosis, the Random Forest performed the best performance with achieving accuracy rates of 97.25% and 98.53% respectively. In contrast, the Convolutional Neural Networks (CNN) performed well in brain conditions and breast conditions with accuracy of 98.17% and 95.13% respectively. This means that different diseases required tailored algorithm to achieve the high accuracy in prediction.

Moreover, Rajora *et al*. (2021) presented a web-based disease prediction system by using machine learning. The users can select the symptoms from the given list for disease diagnosis. The authors also proposed an ensemble voting algorithm to provide the best disease prediction results. The selected algorithms include K-Nearest Neighbor (KNN), Naïve Bayes and Random Forest which combined together as an ensemble approach. As a result, the Random Forest achieved 93.65% accuracy, Naïve Bayes is 84.02% and KNN is 93.53%. For the ensemble model, it fitted to the best model, which is Random Forest, 93.65%.

In addition, Gupta *et al*. (2024), used machine learning classifiers including Random Forest, K-Nearest Neighbour, Logistic Regression (LR), Decision Tree, Multi-Layer Perceptron (MLP), Support Vector Machine and AdaBoost to find out the best model to implement in real-life. There are 4 diseases including in this disease prediction research, which are asthma, diabetes, liver disease, and kidney disease. This research paper used 4 different datasets to predict different disease and observed the performance metrics of different models. In the experiment results, the Random Forest performed best on each dataset with an average accuracy score of 95.8% for kidney disease, 87.83% for diabetes, 95.83% for asthma disease and 99.68% for liver disease.

Table 2.1: Related Work

Title	**Author, year**	**ML algorithms used**	**Dataset**	**Evaluation Metrics**
Revolutionizing Healthcare: Screening system to identify Diseases using Machine learning approach.	Sangeetha. V *et al.* (2024)	Decision Tree, Naive Bayes, Random Forest	Columbia website (132 symptoms and 40 diseases)	Accuracy for DT = 97.53% Accuracy for Random Forest = 95.66% Accuracy for Naïve Bayes = 93.75%
Significance of Visible Non-Invasive Risk Attributes for the Initial Prediction of Heart Disease Using Different Machine Learning Techniques.	Ansarullah et al. (2022)	Decision Tree, K-Nearest Neighbor, Support Vector Machine, Random Forest and Naïve Bayes.	collected from different heterogeneous data sources of Kashmir (India) through quantitative data collection methods (5776 records)	Random Forest - accuracy of 84%, a sensitivity of 85%, a specificity of 83%, an error rate of only 13%, and a precision of 85%.
Multiple Disease Prediction Using Machine	George *et al.* (2024)	Random Forest, Support Vector	4 different datasets (Heart disease dataset, brain	Random Forest achieving accuracy rates

Learning with Chatbot and Doctor-Patient Appointment System.		Machine (SVM), K-Nearest Neighbor (KNN) and Convolutional Neural Networks (CNN)	tumor dataset, breast cancer dataset, Chronic Kidney Disease (CKD) dataset)	of 97.25% and 98.53% respectively. Convolutional Neural Networks (CNN) in brain conditions and breast conditions with accuracy of 98.17% and 95.13% respectively.
Web based disease prediction and recommender system	Rajora, H *et al*. (2021)	Naïve Bayes, Random Forest and K-Nearest Neighbor	dataset from National Centre of Disease Control (NCDC) (4921 unique entries)	Accuracy of Random Forest is 93.65%, Naïve Bayes is 84.02% and KNN is 93.53%.
An Experimental Analysis of Multiple Disease Prediction Using Machine Learning Algorithms	Gupta *et al*. (2024)	Random Forest, Logistic Regression (LR), Decision Tree, Multi-Layer Perceptron	4 different datasets (Diabetes, Kidney Disease, Liver Disease and Asthma)	Random Forest - accuracy score of 95.8% for kidney disease, 87.83% for diabetes,

		(MLP), AdaBoost, Support Vector Machine and K-Nearest Neighbour		95.83% for asthma disease and 99.68% for liver disease.

2.2.2 Compare Existing Models

There are various algorithms have been used to be train, and each of them having its advantages and limitations. There are 5 Machine Learning models are commonly used for disease prediction:

2.2.2.1 Decision Tree (DT)

Decision Tree is one of the most commonly used for supervised learning algorithm, used for both regression and classification tasks (Matzavela and Alepis, 2021). Moreover, Decision Tree is a tree-like structured classifier that starts with a single node representing the attribute tests and branch representing the attribute values and the leaf nodes will represent the possible outcomes (Kosarkar *et al*., 2022). Figure 2.1 shows the example of a Decision Tree and how the DT divide branches and produce possible results.

The strengths of Decision Tree are easy to use and can be learned quickly (Blockeel *et al*., 2023). The training time complexity for Decision Tree is O(n*log(n)*m). Moreover, it is capable of processing both qualitative and quantitative data types. However, Decision Tree usually performs worse when learning from the raw data such as text or sound. This may affect the prediction process since the features for prediction have to be constructed. Decision Tree also easily overfitted and sensitive to the small data changes and lead to totally different trees (*Decision Tree Method: Applications, Pros & Cons, Examples*, n.d.).

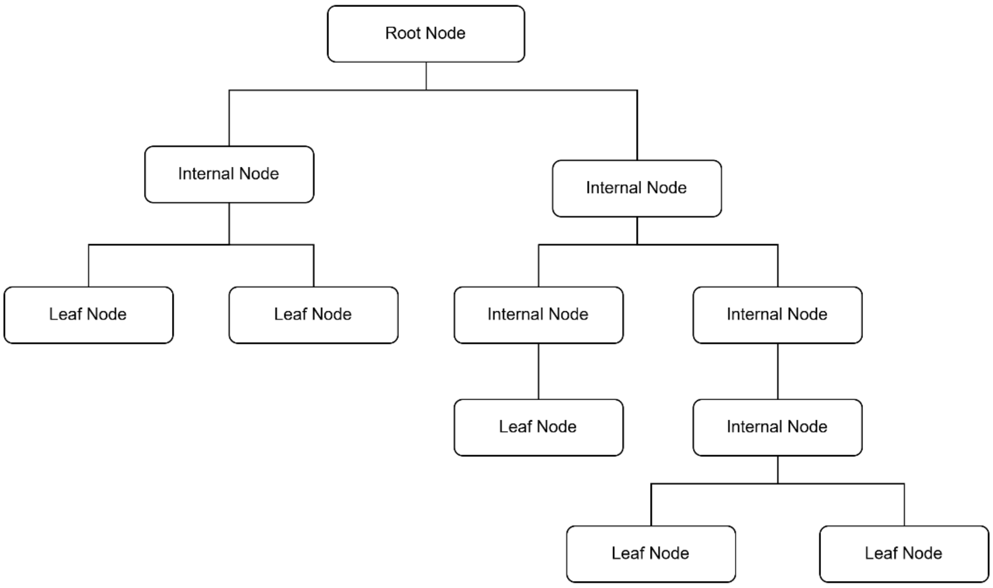


Figure 2.1: Decision Tree Structure Diagram.

2.2.2.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a type of supervised machine learning method commonly applied to classification tasks. (Dey *et al*., 2018). Vapnik introduced SVM as a kernel-driven model designed to handle both classification and regression problems in machine learning. (Cervantes *et al*., 2020). Kernal is a function that maps data to a high-dimensional space, enabling SVM to process non-linearly separable data (Jain, 2024). There have two types of SVM which are linear SVM and non-linear SVM. Figure 2.2 shows an example of a non-linear SVM as a kernel output in a 3-dimensional feature space.

The advantages of SVM are effectiveness in handling high-dimensional space, therefore making it well-suited for datasets with a wide range of features. SVM is capable of managing both linear and non-linear datasets by utilizing kernel functions. According to Gomathy *et. al*., the accuracy score of SVM is 96.49% in predicting the diseases from patient symptoms. However, the computational complexity of training an SVM typically ranges from $O(n^2)$ to $O(n^3)$, where n represents the size of the training dataset. This show that SVM may be inefficient for managing large-scale datasets since it may require more time for training.

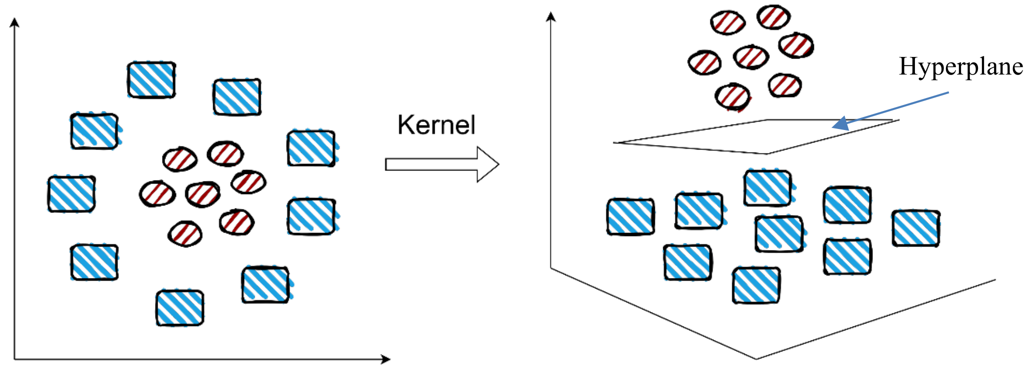


Figure 2.2: Support Vector Machine (SVM) Diagram.

2.2.2.3 Random Forest (RF)

Random Forest (RF) is a type of supervised learning algorithm and is an ensemble classifier that build a group of separate and non-identical decision tree based on the idea of randomization (Ren *et al*., 2017). According to Srihith *et al*. (2023), Random Forest employs an ensemble approach by generating numerous decision trees and combine their output to create more reliable and precise prediction model. They are widely used for regression and classification task. Figure 2.3 show the example of Random Forest (RF) and explains the working of the Random Forest (RF) algorithm.

RF reduces the overfitting relative to use a single decision tree and provides feature importance score to determine which features are more impactful (Srihith *et al*., 2023). Furthermore, RF can handle large numbers of datasets with noise and high dimensionality and overcome the missing value imputation (Zhu, 2020). Zhu (2020) also claimed that the weaknesses of RF are difficult to interpretable result as a single decision tree and difficulty in dealing with high-cardinality categorical variables. According to Song *et al*. (2021), the accuracy of RF is 99.88% in diagnosis pressure ulcer from 19 variables, which is the best prediction performance compared to Decision Tree, SVM, and Naïve Bayes.

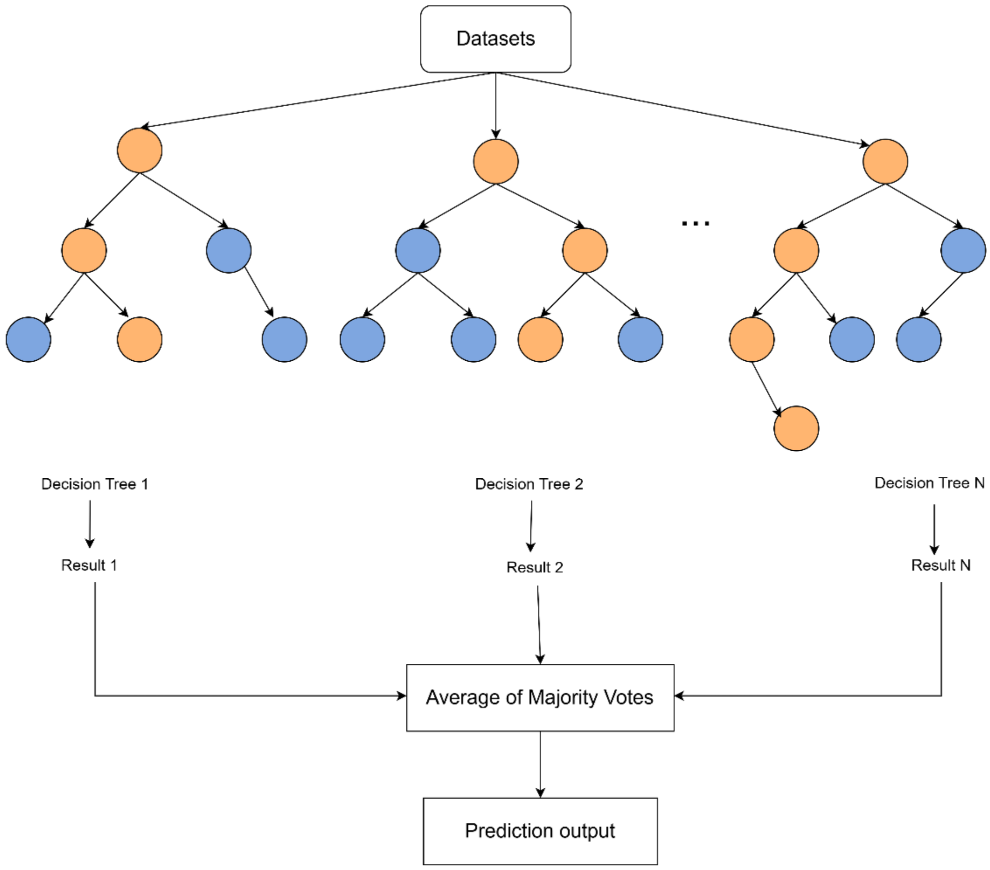


Figure 2.3: Random Forest (RF) algorithm diagram.

2.2.2.4 Naïve Bayes (NB)

Naïve Bayes is a classification technique that operates under the assumption that each value is independent of the other values. This means that a particular feature in a class is not correlated with any other feature (Kosarkar *et al.*, 2022). It is mainly used in text classification. It computes the likelihood of each class based on the observed features and selects the highest probability of the class as the prediction when given new data point (Alahmar and *et al.*, 2023).

The advantages of NB are ease of use and efficiency. NB can work well with the high-dimensional data such as text classification without causing a large computational burden (Beslin Pajila *et al.*, 2023). The limitations of NB are that the feature independence assumption rarely holds in real world data and its effects precision. This could produce undesirable results, especially if the attributes are closely linked or interact with each other in complicated and intricate ways (Beslin Pajila *et al.*, 2023). For instance, the symptoms for cough and fever are often correlated in disease prediction.

2.2.2.5 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a straightforward and effective algorithm that classifies the new data points by comparing them to the most similar classes as known as nearest neighbors (N) (Harish *et al*., 2021). The new data can be quickly categorized into appropriate classes when it first appears by using the KNN technique. KNN also known as a lazy learning algorithm because it stores the entire training dataset and perform computations at prediction time rather than learning from it instantly (Sreedevi *et al*., 2022). Figure 2.4 shows the example of K-Nearest Neighbors (KNN) algorithm working visualization.

The strengths of KNN are that it is a simple to implement algorithm for solving problems and it is very resistant and tolerant to the noise that prevailing in the training dataset (Bansal *et al*., 2022). For the KNN weaknesses, Bansal *et al*., 2022 stated that KNN determine the appropriate value of K is complex because it can sometimes dramatically change the results. In addition, KNN prediction stage is slower for larger dataset, and it requires more storage space compared to an effective classifier (Taunk *et al*., 2019).

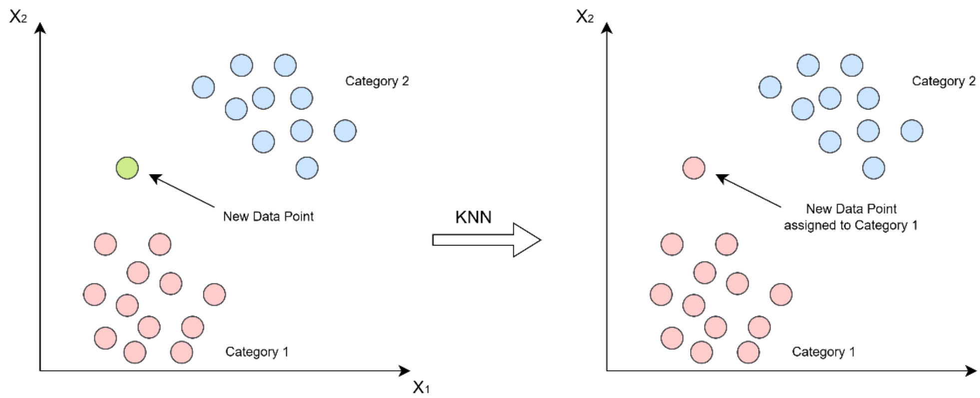


Figure 2.4: K-Nearest Neighbors (KNN) diagram.

2.2.3 Summary of Comparative Models

All of these 5 models are classification, which is suitable for disease predictions.

Table 2.2: Comparison between DT, SVM, RF, KNN, and NB.

Model Criteria	Decision Tree	Support Vector Machine (SVM)	Random Forest	K-Nearest Neighbors (KNN)	Naïve Bayes
Best Use Case	Handles both categorical and numerical data	High-dimensional data	Robust classification, feature importance	Irregular decision boundaries	Text-based or independent features
Interpretability	High	Low	Medium	Medium	Medium
Training Speed	Fast	Slow	Moderate	Very fast	Very fast
Prediction Speed	Fast	Slow	Moderate	Slow	Very fast
Handles Noisy Data	Moderate	High	High	Low	High
Handles High Dimensions	Moderate	High	High	Low	High
Data Size Requirement	Small to medium	Small to medium	Medium to large	Small to medium	Small to medium
Overfitting Risk	High	Low	Low	Medium	Low

Performance in Disease Prediction	Good for interpretable tasks, but may overfit	High accuracy in high-dimensional data	Robust, often high accuracy in medical tasks	Good for simple datasets, struggles with high dimensions	High accuracy

According on the comparison table above, the Decision Tree is good for handling the numerical and categorical data, and its interpretability is high, but it has high risk of overfit which can lead to incorrect disease prediction results compared to SVM, RF, and NB. Although SVM can handle high dimension data well but the interpretability and training speed of it is quite low and may not be suitable for disease predictions system as the system also required fast prediction speed. Compared to SVM, the KNN is very fast to trained since it skips the training phase and can simply stores the data. However, the prediction time of KNN is slow, the same with SVM, it may decrease the user experience as the waiting time for results is longer. Naïve Bayes is very fast for the training and predicting time, and the overfit risk is low. However, the accuracy of Naïve Bayes may be affected since the independent assumptions is often violated in medical data. The RT has not the limitations of independent assumptions constraints and can handle noisy data and high dimensional data well. Although the prediction speed of it is moderate, but also acceptable in web applications, and it can simply implement. Overall, the RT shows the best characteristics in disease predictions.

2.3 Compare Existing Web App

There are a number of existing disease prediction web applications that can identify potential diseases based on user-inputted symptoms. Studying these similar web applications provide valuable insights of the functionality and requirements of the disease prediction system, which can help to identify the gaps and define the strengths and limitations of the existing disease prediction web applications. This section compares three existing web applications which are Symptomate, WebMD Symptom Checker, and Your.MD (Healthily).

2.3.1 Symptomate

The Symptomate is an Artificial Intelligence tool for symptom checker developed by Infermedica in 2012. This symptom checker will analyse the user's symptoms, predict the diseases the user may be suffering from and provide some recommendations to the user for further actions. This system allows user to type and select the symptoms. The introduction of Symptomate is accepted with the terms of service and agrees to the privacy policy. The Symptomate has an interview for users before they enter their symptoms. The interview questions such as survey respondent, age, gender, and some user health conditions. Figure 2.5 shows the Symptomate survey respondent's page for user to select. The interface of Symptomate is clear and intuitive.

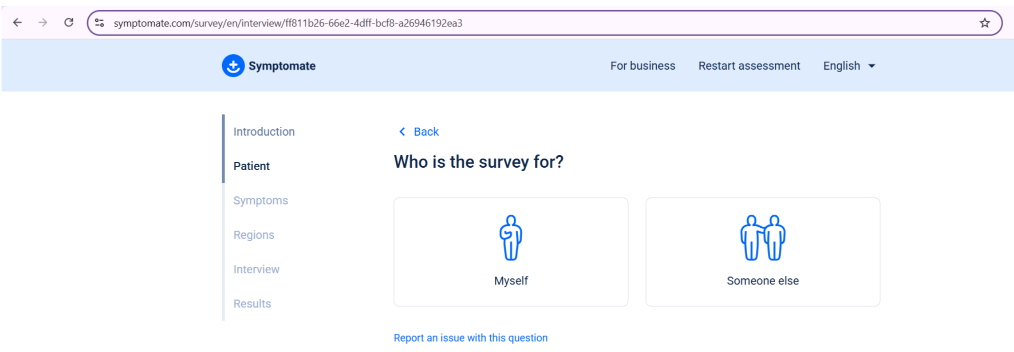


Figure 2.5: Symptomate Survey Respondent's Page.

There are a few statements that need to be answered by the user to gain deeper insights into the user's health conditions before starting the interview about the user's symptoms. After that, user can start to enter their symptoms. When the user enters the symptoms, it may display some relevant symptoms for the user to select. The checklist-based symptom entry system is used to minimize errors by providing predefined options, making it accessible to users unfamiliar with medical terminology. The user can add multiple symptoms for more accurate assessment. Figure 2.6 shows the User Input Symptoms page for user to select and add symptoms they have. The checklist input function may limit the natural language understanding. The user cannot describe the symptoms using their word like a sentence.

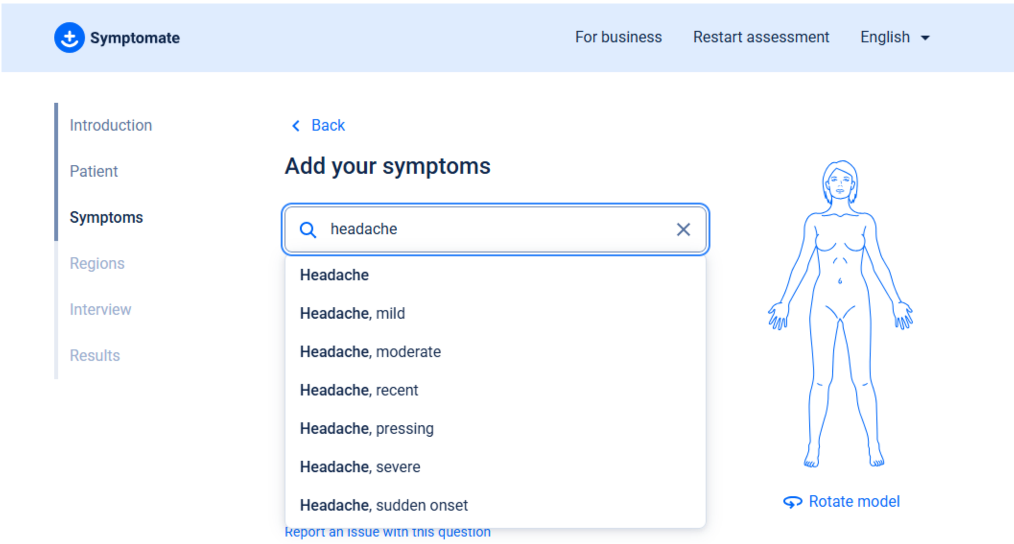


Figure 2.6: Symptomate User Input Symptoms Page

In order to further diagnose the disease, the system asks several questions and symptoms the user may have. Figure 2.7 shows the example of a multi-select symptoms interview question page. The system allows the user to enter more than one answer to this question. This helps the user save time by clicking on multiple answers instead of typing them in one by one themselves.

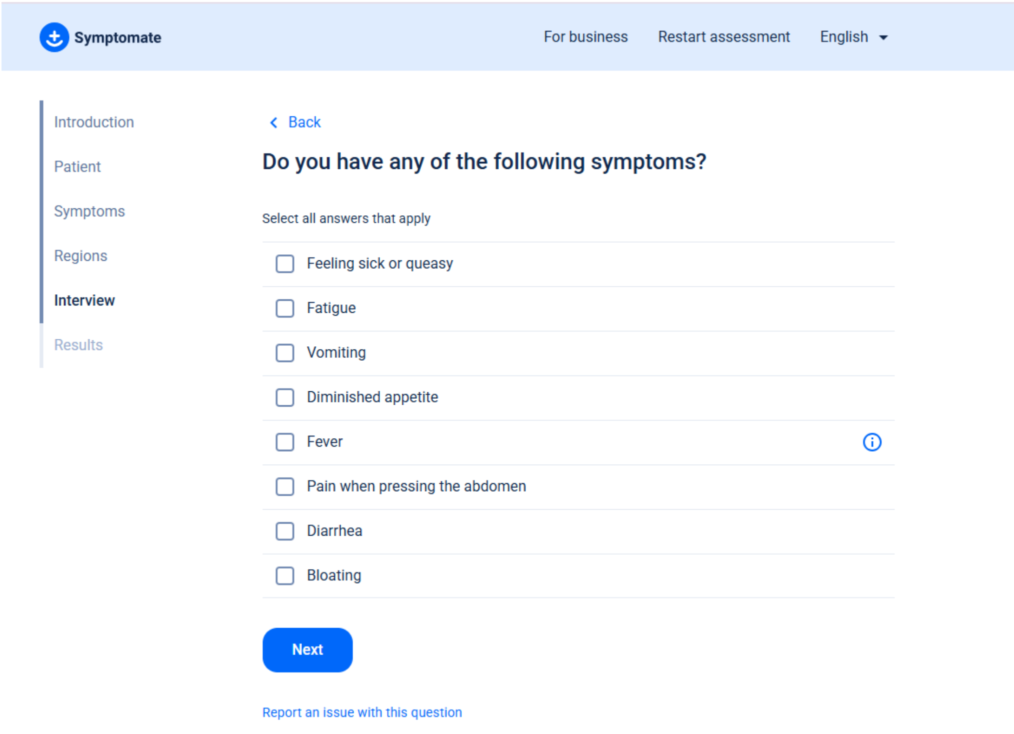


Figure 2.7: Example of a Multi-Select Symptoms Interview Question Page.

After that, the Symptomate system provides the potential diseases that the user may have. In addition, Symptomate will provide some suggestions for the potential diseases. If the underlying disease is mild, the system will suggest that self-care is sufficient. Figure 2.8 shows the example of predicted result for a mild disease. The system will also provide a variety of possible conditions based on the symptoms, rather that providing a disease prediction result for just one disease. The blue 'Show common care method' button provides some information and details about the predicted disease. This provides user with more information and have better understanding about the potential diseases.

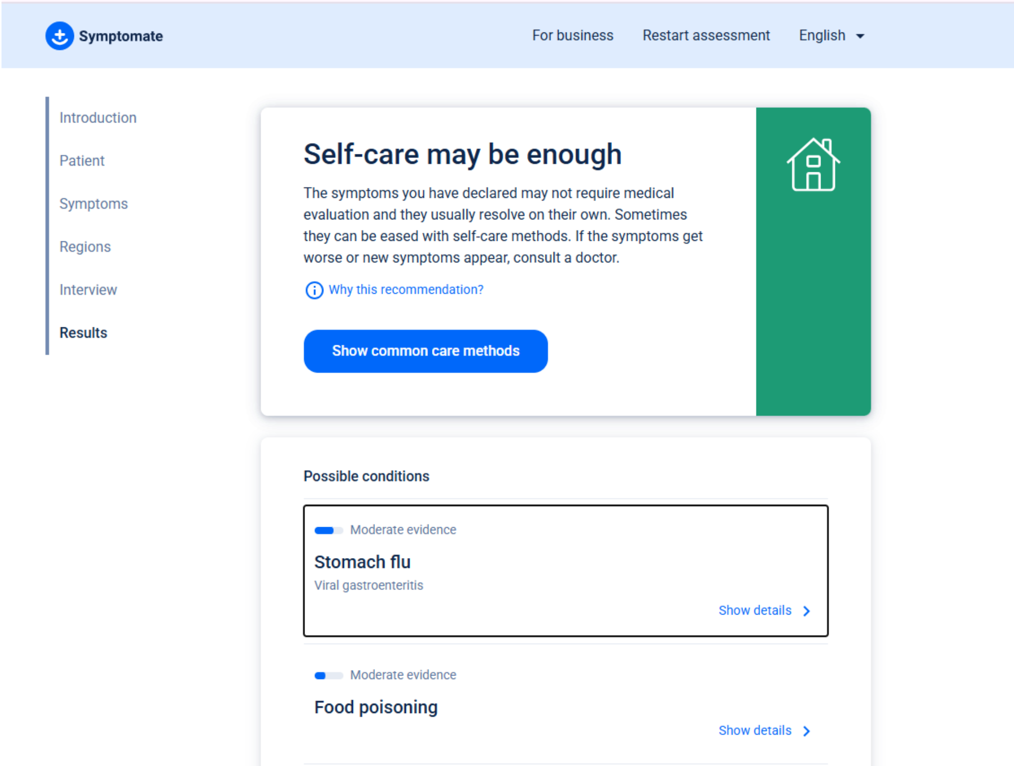


Figure 2.8: Example of Predicted Result.

The UI and UX of Symptomate system are simple and clear, allowing users to easily enter or select the symptoms. According to a study by BMJ Open, the top 3 diagnostic accuracy of Symptomate were 27.5%, which is lower than the general practitioners (GP)s' 82.1%. However, the urgency advice safety was high at 97.8%. Symptomate lacks the flexibility of free text input, which would better capture a wide range of symptom descriptions.

2.3.2 WebMD Symptom Checker

WebMD Symptom Checker is one of the well-known online web applications for early diseases diagnosis. It provides a comprehensive interface for users to select symptoms by body location and input symptoms to obtain the possible conditions. Same with Symptomate, the WebMD Symptom Checker likewise prompts the user to provide their age and gender details. Figure 2.9 shows the example of WebMD Symptom Checker Info Page which requires user to enter age and sex before starting to enter their symptoms. The interface of WebMD Symptom Checker is clear but not centred enough to grab the user's attention in the first place.

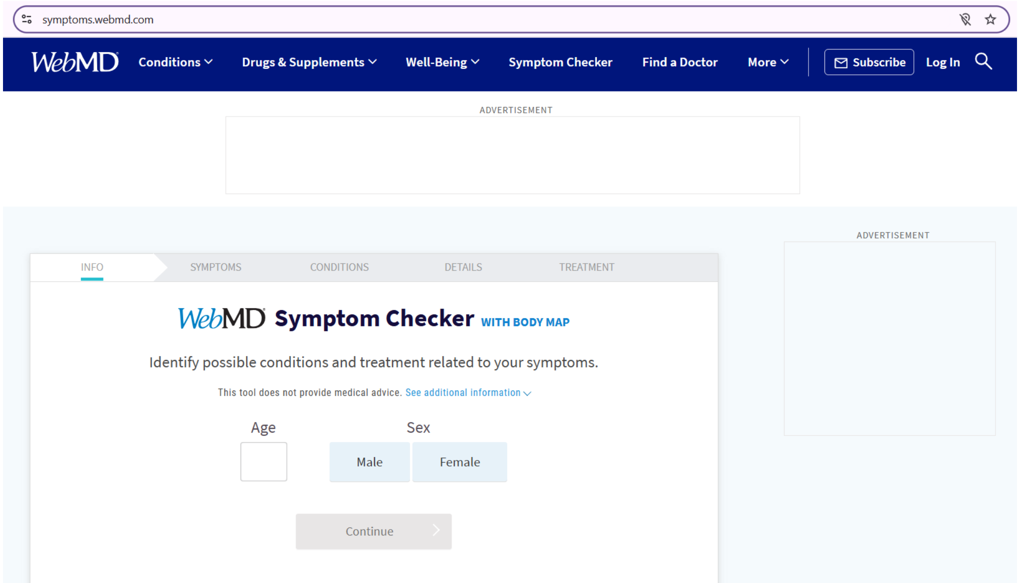


Figure 2.9: Example of WebMD Symptom Checker Info Page.

In the Input Symptoms page, the user can enter their symptoms and select the appropriate symptoms from the drop-down list. The drop-down list will only display the symptoms that are related to the symptoms entered by user. The user can choose from a wide range of symptoms by entering a keyword simply. Figure 2.10 shows a drop-down list of symptoms that related to skin. The user can also tap on body parts to select symptoms by body location. This is more convincing if the user does not know how to describe the symptoms in text. The Figure 2.11 shows the WebMD Input Symptom Page with symptoms selected by body location. This visualization method works well for users who don't know how to express symptoms, improving accessibility and user engagement. The selected symptoms will be displayed at the bottom part.

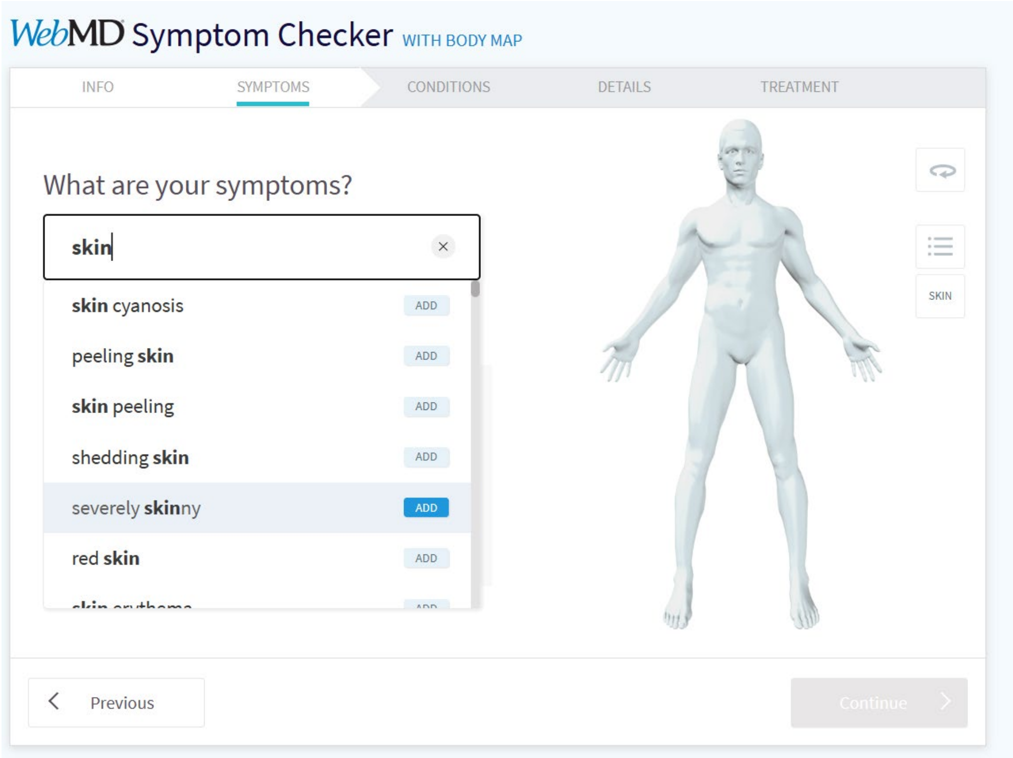


Figure 2.10: Example of WebMD Input Symptom Page.

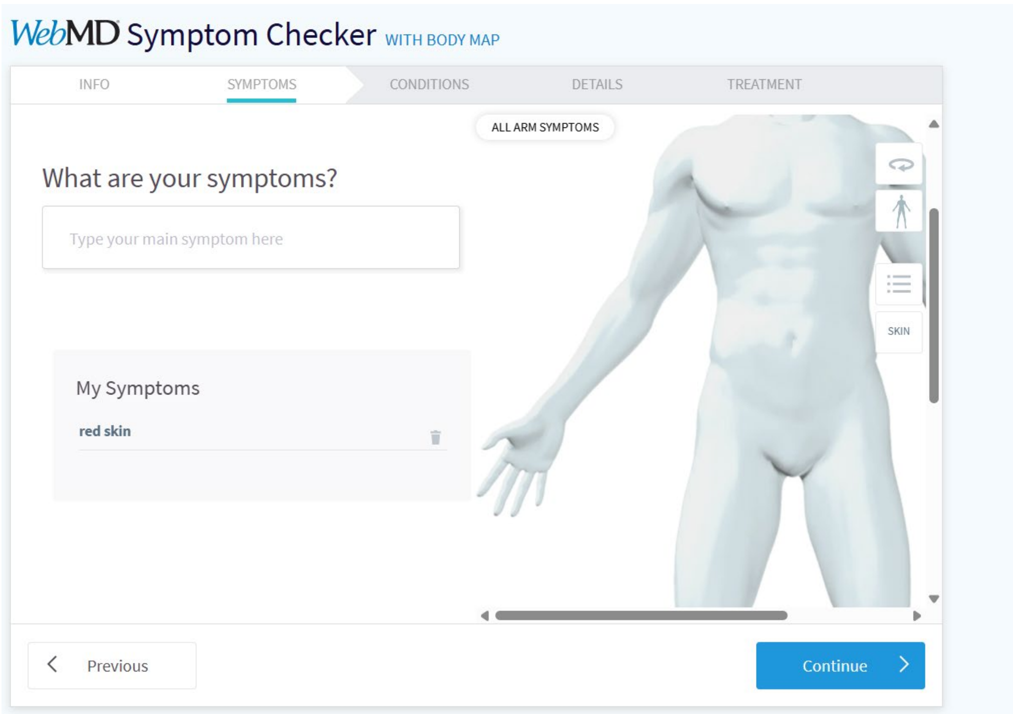


Figure 2.11: Example of WebMD Input Symptom Page for Select Symptoms by Body Location.

After adding all the symptoms, the WebMD Symptom Checker system starts to analyse and provide the potential conditions that match to the symptoms. The system provides the potential diseases and ranks them in order of strong matches up to the fair matches. A strong match indicates that the user may have a high level of potential diseases, while a fair match is the opposite. The left-hand side of the web page displays the disease that the user may suffer from, and the right-hand side shows the user's symptoms the details. Figure 2.12 shows the example of the conditions page. The WebMD Symptom Checker allows user to modify their age, gender and symptoms to start over the prediction process. This provides users with a more flexible method of continuing to predict diseases, even if they enter incorrect information.

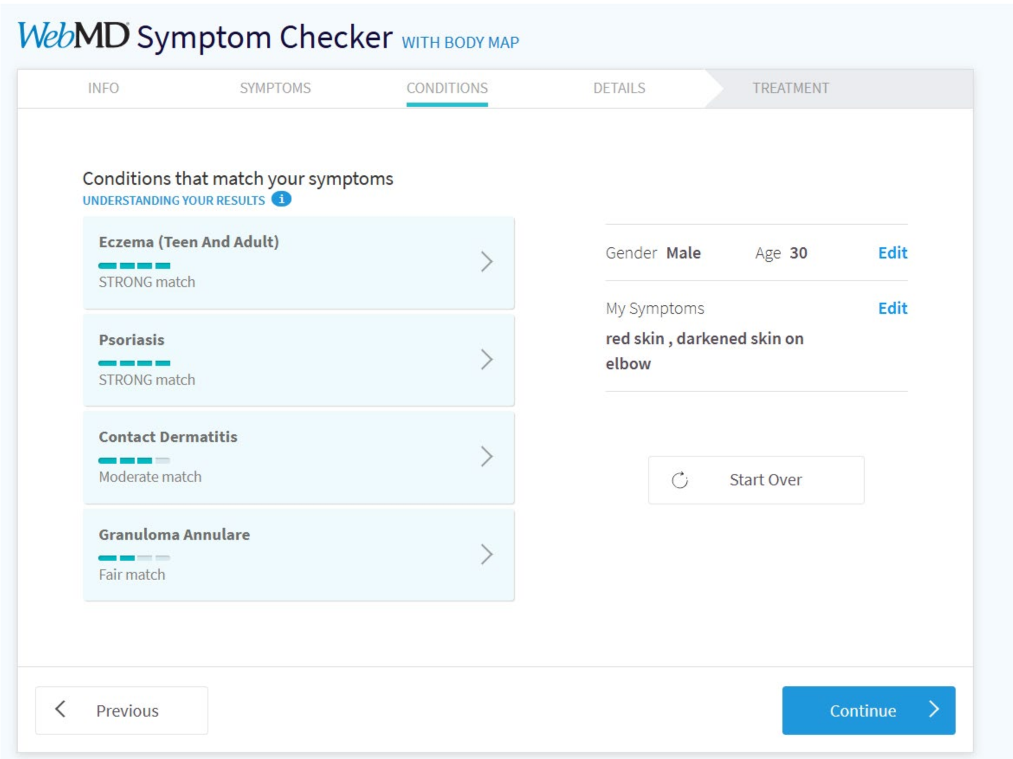


Figure 2.12: Example of the Conditions Page.

The WebMD Symptom Checker display detailed information about each disease for the user to better understand the disease. Figure 2.13 shows an example of the Condition Details page, which contain the relevant information for the disease. Moreover, the system also provides the treatment options for user to take further action.

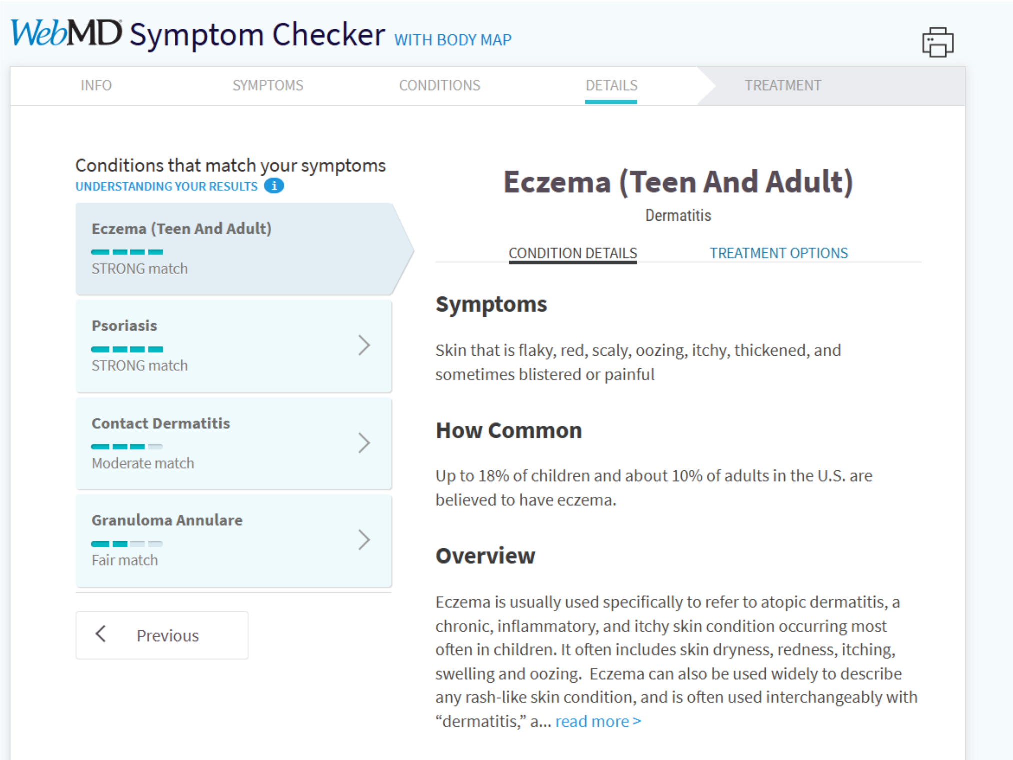


Figure 2.13: Example of the Conditions Details Page.

WebMD Symptom Checker provide a straightforward and appealing visual approach. The symptoms input methods of WebMD Symptom Checker are drop-based on a down list of checklists for user to select the appropriate symptoms or by clicking on body parts. A study by BMJ Open claimed that the top 3 suggestion accuracy rate of WebMD Symptom Checker is 35.5%, which is lower than 82.1% for general practitioners (GPs) also. This indicates that a moderate level of reliability for WebMD Symptom Checker. The system lacks a follow-up interview process for refinement of predictions, which may lead to overly broad results, especially if symptoms overlap.

2.3.3 Your.MD (Healthily)

Your.MD also known as Healthily, is a web-based symptom checker that uses a chatbot to help users to identify the potential illnesses. The Healthily utilized AI and chatbot system to analyse the symptoms and provide the user with guidance to the next steps based on their symptoms. The user of Healthily must be at least 16 years old and agree to the Privacy Policy before using the system. Same with the WebMD Symptom Checker and Symptomate, the system requires user to enter their gender and year of birth. Figure 2.14 shows the Healthily Gender Page, where user to select their gender. The UI of Healthily is simple and clear.

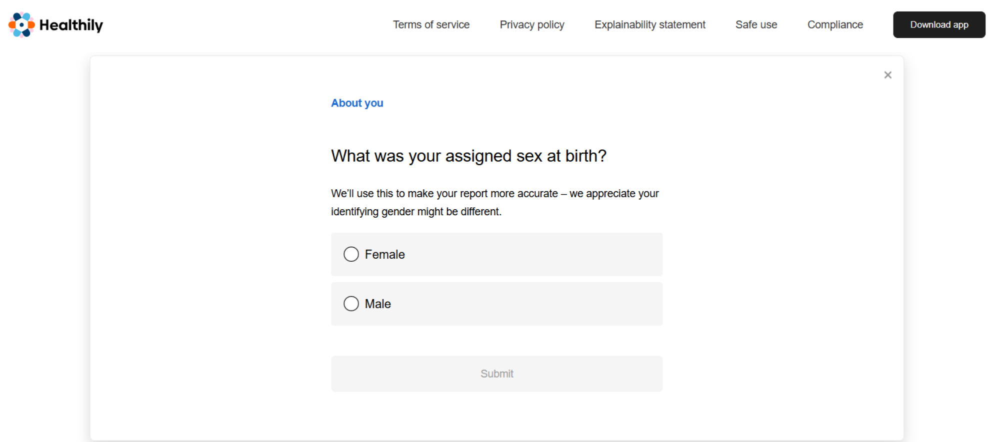


Figure 2.14: Healthily Gender Page.

Figure 2.15 shows the chatbot page of Healthily, where user can enter their symptoms in words or sentence. After submitting, the Healthily system provides the user with relevant symptoms that they may have. The user can select which symptoms they may have by clicking on the checkbox of the symptom, and the Healthily also allows user to add additional symptoms later. Its conversational interface is a major advantage, allowing users to enter symptoms in natural language, which the system can maps to predefined symptoms. Figure 2.16 shows the example of Select Symptoms page for the user to check the symptoms they may have. The symptoms provided in Healthily are extracted from the words or sentences entered by the user. If none of these symptoms, the Healthily will ask user to enter more specific symptoms or reword the symptoms. The system will then provide more options for user to choose from. The user can select one or more options if the symptoms appeared. This flexibility caters to users who prefer to describe symptoms in their own words and increasing inclusivity.

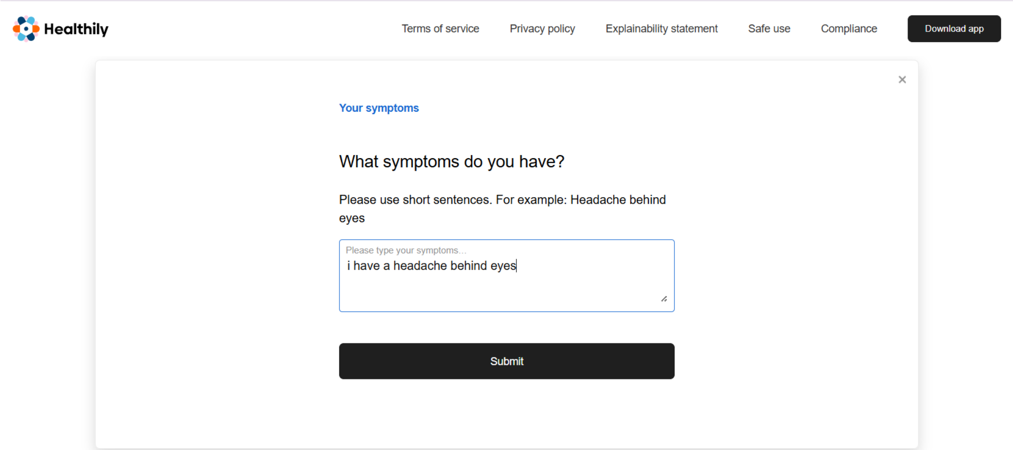


Figure 2.15: Healthily Chatbot Page.

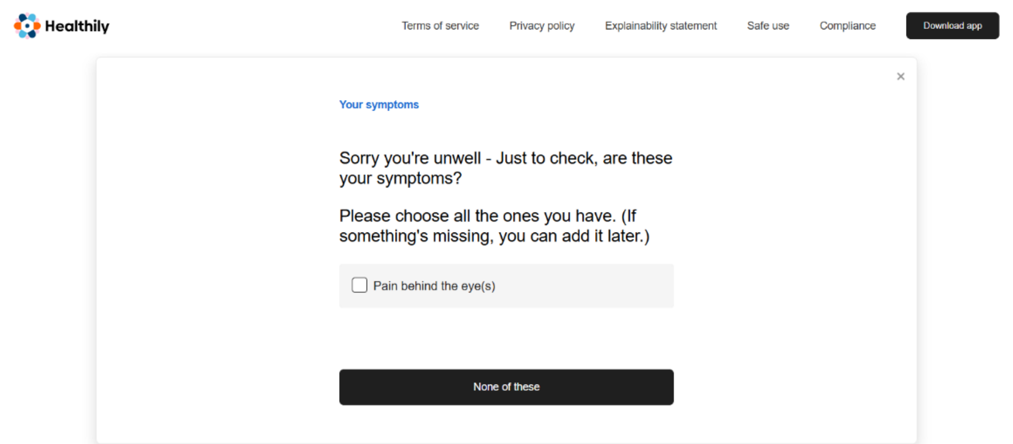


Figure 2.16: Healthily Select Symptom Page.

After selecting the relevant symptoms, Healthily also asks follow-up questions about symptom duration and health context (e.g., history of diabetes) to refine the prediction by taking time and medical context into account. Figure 2.17 shows symptom checker page that prompting user to indicate the duration of symptoms. For health background question, the user can select one or more options. Figure 2.18 shows the Healthily prompting the user to indicate the health background.

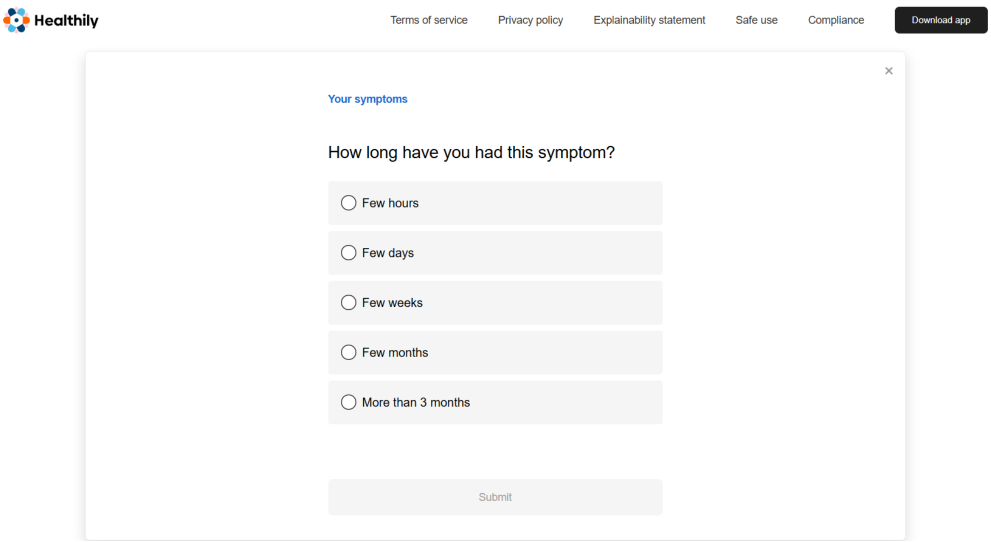


Figure 2.17: Healthily Symptom Checker Page prompting user to indicate the duration of symptoms.

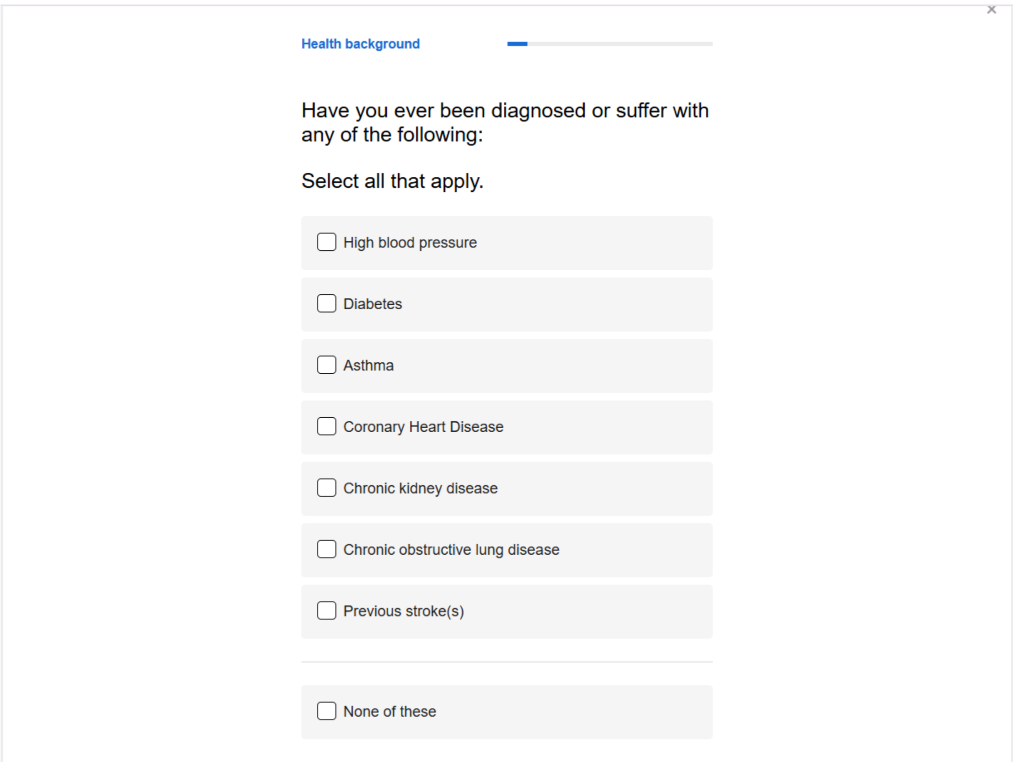


Figure 2.18: Healthily prompting user to indicate the health background.

Furthermore, the Healthily provides some related symptoms that the user may be experiencing. This can ensure that the more accurate disease prediction results are provided. However, the process of Healthily is overly lengthy and the multiple questions stages can be frustrating for users looking for a quick assessment. After answering all the questions, the Healthily generates a prediction report for user based on the symptoms. Figure 2.19 shows an example of prediction report page which include the possible causes and the summary of symptoms. The user can click on the possible cause to access the detail information about the possible disease. The summary displays the symptoms user selected by user in the previous questions, and the system provide suggestions for further action to be taken by the user. User are provided with a clear and straightforward understanding of the possible causes of the diseases.

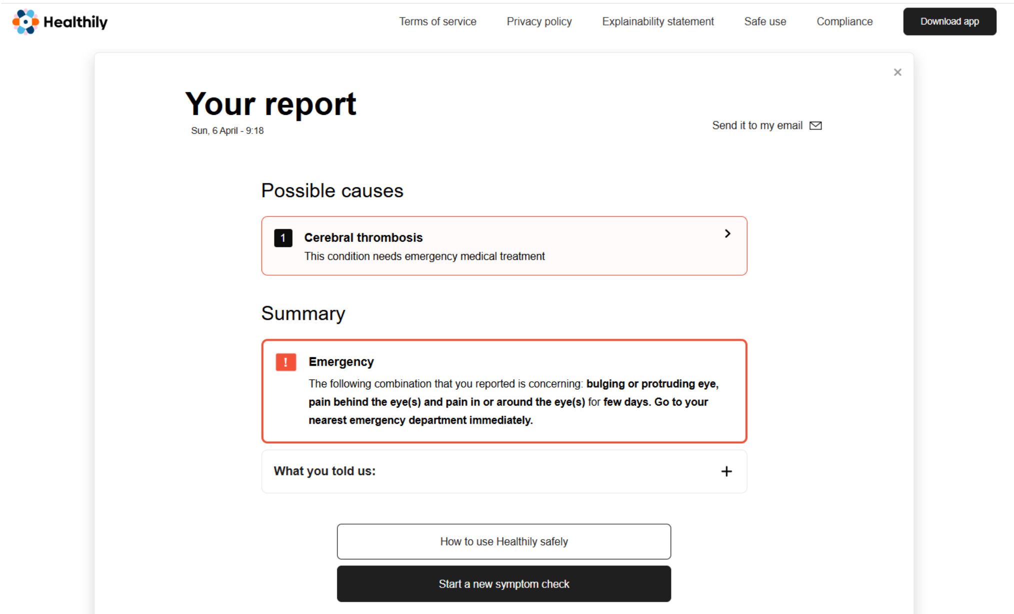


Figure 2.19: Healthily Prediction Report Page.

The Healthily provide a conversational chatbot interface that allow user to enter symptoms with intuitive questions. However, the process of it is lengthy and overloaded of information. This may not be suitable for users who want to get a quick assessment for disease prediction. The system also offers some suggestions to the user after the prediction results are available. A study by BMJ Open stated that the top 3 suggestion accuracy rate of Your.MD (Healthily) is 23.5%, which is lower than 82.1% for general practitioners (GPs) also. This indicates that a moderate level of reliability for Your.MD (Healthily).

2.3.4 Summary of Existing Applications

Table 2.3: Comparison between Symptomate, WebMD Symptom Checker, and Your.MD (Healthily).

Feature	Symptomate	WebMD Symptom Checker	Your.MD (Healthily)
Platform	Web, Mobile	Web, Mobile	Web, Mobile
UI/UX	Clear but lengthy questions set	Interactive body maps with some accessibility issues	Conversational chatbot but lengthy
User Input Method	Checklist	Body map and dropdown menu	Chatbot with free text
Personalization	Basic (age, gender)	Basic (age, gender)	Advanced (age, gender, lifestyle, risk factors)
Diagnosis Output	List of possible conditions	List of possible conditions	Suggested disease condition with self-care advice
Accuracy	27.5% (BMJ Open)	35.5% (BMJ Open)	23.5% (BMJ Open)

Table 2.2 shows the comparison between Symptomate, WebMD Symptom Checker, and Your.MD (Healthily). Based on the table above, each of these three existing web applications has its own advantages and disadvantages. The usability for Symptomate and Your.MD (Healthily) are clear and easy to use while WebMD Symptom Checker is more complex with many options. The WebMD Symptom Checker is engaging but the lengthy processes can affect the user experience due to the large number of questions that need to be answered. According to BMJ Open, the accuracy of these applications is quite low, and this may provide user with some inaccurate results. The diagnosis output of

Your.MD (Healthily) is quite good, providing the suggested disease condition and self-care advice that the user can take some action on after diagnosis.

2.4 Evaluation Metrics

Evaluating metrics is crucial for assessing the effectiveness of machine learning models, especially in medical diagnostics. The evaluation metrics helps to determine the usability and reliability of the models. There are a number of evaluation metrics that can evaluate the effectiveness of machine learning algorithms, such as sensitivity, accuracy, F1-Score, precision and specificity. The diagnosis results can be categorized as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP and TN signify an accurate model prediction, whereas FP and FN denote an incorrect model diagnosis.

2.4.1 Accuracy

Accuracy refers to the ration of accurate predictions cases to the overall number of cases. This evaluation metric is often utilized to access the performance of machine learning models, especially in tasks of classification and defect detection (Ashfakul Karim Kausik *et al.*, 2025). High accuracy suggests that the models is reliable, and the prediction results are more accurate and credible. The advantages of accuracy are simple and easy to use. Accuracy is reliable for balanced datasets. However, it can be misleading when applied to the imbalanced datasets.

The equation of accuracy is expressed as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

2.4.2 Sensitivity (Recall)

Sensitivity also known as Recall, is measured by the ration of true positive cases accurately identified. Recall measures the capability of models to identify the faults in the dataset correctly (Ashfakul Karim Kausik *et al.*, 2025). High sensitivity denotes that the model able to identify most true cases and reduce the cases of miss diagnoses because false negatives can lead to serious consequences. It is important to detect the potential faults in the models and intervene early.

The formula of sensitivity is:

$$Sensitivity = \frac{TP}{TP + FN}$$

2.4.3 Specificity

Specificity is measured by the ratio of true negative cases correctly detected. This metrics reflects the ability of models to identify the true negatives in each available category. For example, specificity refers to recognizing that the patient does not have a particular disease. It is important to avoid the unnecessary treatments in the disease prediction. The specificity can be computed by dividing number of true negatives by the total of true negative and false positive.

The formula of specificity is:

$$Specificity = \frac{TN}{TN + FP}$$

2.4.4 Precision

Precision is measured by the ratio of precisely identified positive cases among the total number of expected positive cases. The precision indicates the reliability of a positive prediction. This is important when the false positives can cause high costs such as unnecessary testing. High precision ensures that the model predicts correctly. The formula of precision is calculated by dividing the number of true positive by the total of the true positive and false positive.

The formula of precision is:

$$Precision = \frac{TP}{TP + FP}$$

2.4.5 F1-Score (F-Measure)

The F1-Score, also known as F-Measure, represents the harmonic average of precision and sensitivity (recall). It provides an equitable view of the capability of the model to minimize both false positive and false negative. This is important to detect as many true cases as possible meanwhile ensuring high precision. The ranges of F1-Score are between 0 and 1, where 1 indicates the best sensitivity and precision.

The formula of F1-Score is:

$$F1 - Score = 2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity}$$

2.4.6 Summary of Evaluation Metrics

Table 2.4 Comparison of different evaluation metrics.

	Advantages	**Disadvantages**
Accuracy	Simple, reliable for balanced datasets.	Can be misleading in imbalanced datasets.
Sensitivity (Recall)	Detect true cases, minimize the false negatives	May increase false positives
Specificity	Reduces false positives, avoid unnecessary alarm.	May increase false negatives, miss actual cases.
Precision	Reliable positive predictions, useful when false positive is costly	May reduce sensitivity, miss true cases
F1-Score (F-Measure)	Balances precision and sensitivity, good for imbalanced data.	True negatives were not considered.

Based on the table 2.2, the accuracy is simple and contains strong overall metric due to the balanced dataset, but it unable to distinguish between false positives and false negatives, which is critical in healthcare. It is useful for comparing models but not sufficient as the primary metric. Specificity is important to prevent unnecessary alert, such as such as false predictions of disease, causing stress and leading to unnecessary treatment or tests. However, specificity may increase the false negatives and be more harmful in healthcare. For precision, it can ensure that the model correctly predicts disease, but prioritizing precision may lead to reduce in sensitivity, cause more false negatives. In addition, F1-Score can balance sensitivity and precision, it can ensure the prediction are both reliable and comprehensive. However, F1-Score is not considered the true negatives. Sensitivity is the most suitable for the disease prediction. This is because it can minimize the false negatives such as missed diagnoses. A missed diagnosis may lead to delay in early treatment with serious consequences.

Although sensitivity may increase the false positives, but it is more acceptable than false negatives, because it is better to incorrectly predict disease than to miss a diagnosis.

2.5 Dataset Sources

Sangeetha *et al*., 2024, used the dataset collected from the Columbia website as the data sources in the project. The dataset included 132 symptoms and 40 diseases. There are several symptoms used to cover 32 diseases in their project. For the diseases cover in the project included normal fungal infection to typhoid diseases. They pre-processed the raw data to overcome the problems of inconsistent data, missing data and noisy of raw data in the dataset. By using the processed data, the authors trained and tested the models in a ratio of 80 to 20.

The dataset collected by Ansarullah *et al.* (2022) for risk modelling was derived from an innocation non-invasive heart disease dataset containing 5776 entries from various heterogeneous data sources in Kashmir (India). Of these 5776 records, 47.5% or 2747 had heart disease, and the other 52.5% or 3031 were in good health. The authors performed a class balance assessment since the heart disease database contain significantly imbalanced data that can lead to bias in machine learning algorithms.

In addition, George *et al*., 2024 used diverse dataset for multiple disease predictions to exam the accuracy of various algorithms. The authors used 4 databases to predict the heart disease, which are Cleveland, Hungary, Switzerland, and Long Beach V. The databases contain 76 attributes, and it divided into two parts for training and testing datasets. Furthermore, the authors used dataset brain tumour classification MRI dataset for brain tumour diagnosis. The brain tumour database contains 2 classes, YES for having the brain tumour, otherwise is No. The other dataset is UCI Repository for predicting the kidney disease. There are some diagnostic measurements in the dataset for predicting the disease.

The dataset acquired by Rajora *et al*., 2021 was sourced from National Centre of Disease Control (NCDC). The dataset demonstrates the symptoms of potential diseases. The dataset includes detailed survey data and the most frequent symptoms among the patients. There are 4921 unique entries were extracted from the symptoms in the database. Individual entries may contain similar disease that have been identified but have different symptoms across different records for the same disease. Through the refinement of the initial entries, the dataset was organized into another dataset.

There are 4 different datasets used by Gupta *et al*., 2024 to predict 4 different diseases. Firstly, is Asthma dataset which contains a total of 29 columns and 2392 entries. This dataset is used to determine which individuals are most risky to have asthma disease. It includes features such as diet quality, smoking and so on. The second dataset is Diabetes dataset which contain 22 columns and 253,680 entries. The dataset includes features such as blood pressure, cholesterol and so on. The third dataset is liver disease dataset, which contain 11 characteristic and 30,691 instances. The last dataset is kidney dataset which contains 54 characteristic and 1659 entries. This dataset can predict the kidney disease based on the patient's medical problems. All of these datasets were collected at Kaggle.

The Symptom-Disease Prediction Dataset (SDPD) was published by Jay Tucker in 2024. The dataset contains 4920 instances, 132 symptoms features and 41 unique diseases. This dataset is contained a variety of data needed for disease predictions system. The SDPD is a tabular dataset with binary symptoms features such as 0 or 1 represent the absence or presence of symptoms. The dataset is suitable for training and accessing the machine learning algorithms. The instance represents a unique combination of symptoms and the corresponding disease diagnosis.

2.5.1 Summary of Data Source

Table 2.5: Comparison of different data source

	Columbia Dataset	**Kashmir Dataset**	**Heart Disease (CHSLB)**	**NCDC Datasets**	**Kaggle Datasets**	**SDPD**
Data Type	Tabular, mixed (numerical, categorical)	Heterogeneous (tabular, text)	Tabular, mixed (numerical, categorical)	Tabular, mixed (numerical, categorical)	Tabular, CSV format	Tabular, 132 binary symptoms, categorical target
Size and Scope	132 symptoms	5776 records	76 attributes	4921 unique entries	Varies	4,920 instances, 132 symptoms feature
Disease Coverage	40 diseases	heart disease only	heart disease only	Many	4 diseases	41 unique diseases
Quality	High, clinical data	Variable, preprocessing needed	High, some duplicates	High, some biases	High, some biases	High, clean, balanced
Availability	Restricted	Likely restricted	Public	Public, but some restrictions	Public	Public

Most of the data sources are tabular data except for the Kashmir dataset which is heterogeneous. The heart disease dataset (CHSLB) and Kaggle datasets are limited to predict a single disease such as heart disease, which may restrict users form predicting other disease although they are easy to access. However, the Columbia and NCDC datasets are broad and large-scale data, but they are more suited to populations health studies and may have limited access and require permissions. The Kashmir data source is too heterogeneous and region-specific. This may affect the diagnostic results of users in other regions. In addition, the Kashmir data source often require pre-processing before use and are of variable quality. The SDPD is the best suited for the disease prediction because it supports symptom-disease mapping using binary features, covers 41 diseases, and requires minimal preprocessing. The size and scope of the SDPD is also quite large, allowing it to cover more diseases. The quality of it also high and balance.

2.6 Web Application Framework

The web application frameworks are the resources and tools used by developers to develop the online software, manage the websites, and so on (Sheldon, 2023). This goal of this project is to develop a disease prediction web application using machine learning and therefore it is necessary to select the appropriate and right tools and technology stack to ensure the reliability and responsiveness of the system. There are two components need to be considered, which are the backend and frontend. This section is to evaluate these frameworks and compares the strengths and weaknesses of each of them.

2.6.1 Backend Framework

Backend is responsible for handle the server-side request. There are two backend frameworks need to be considered in this part, which are the Flask and Django.

2.6.1.1 Flask

Flask is a minimalistic, and micro web framework for Python, commonly utilizing in small to medium-size applications (GeeksforGeeks, 2023). It allows the developers to select their preferred libraries for additional functionality.

Flask is user-friendly because of its simple design and requiring minimal boilerplate code. Flask also allows easy integration with machine learning models and therefore suitable for machine learning based application.

Without manual configuration, the scalability of Flask is limited. The features such as database configuration and authentication are required additional setup, which can increase the complexity to the system. The performance of Flask is fast for small to medium-scale applications. The Flask is suitable for API-driven applications such as machine learning serves that provide low latency due to its minimal overhead.

Strengths of Flask:

- Simple and flexible. Flask allows developers to customize the structure of the application to fit their needs and make it easy to quickly build the APIs and integrate the machine learning models.
- Lightweight and rapid development. Flask can ensure the fast performance and quick setup due to its small footprint, hence it is suitable for small and medium-sized projects.

Weakness of Flask:

- Limited scalability. Flask requires numerous efforts to scale the large applications, which can complicate future scaling of the application.
- Lacks built-in features. Flask requires manual integration as it lacks built-in tools for specific functionality. This may slow down the development of the system.

2.6.1.2 Django

Django is known as the “contained battery” philosophy (GeeksforGeeks, 2020). This means that Django is suitable for rapid development of web applications without having to consider about planning the application's framework in advance. Django offering the built-in functions such as authentication, ORM and so on, making it suitable for large projects. Djonga has good scalability because it has built-in tools for handling the large-scale applications. The ORM and middleware of Djanga is important for managing the complex features and increased traffic.

Strengths of Django:

- Secure and scalable. Django support large-scale applications and it includes the middleware and authentication such as CSRF protection for security purpose.
- Comprehensive Features. Django includes a variety of built-in tools such as authentication, ORM and other to minimize dependence on external libraries, and support rapid development of complex functionality.

Weakness of Django:

- Complexity. The structure approach of Django and the learning curve can be complex and difficult, which can slow down the initial development process.
- Less flexible. The Django has a very steep learning curve and is therefore less flexible when it comes to customizing workflows.

2.6.1.3 Summary of Backend Framework

Table 2.6: Comparison between Flask and Django

Features	**Flask**	**Django**
Flexibility	High	Moderate
Complexity	Simple and lightweight	Moderate and structure
Scalability	Moderate and require manual setup	High, include built-in tools for large-scale application
Performance	Lightweight and fast	Slightly heavier
Suitability	Small to medium application	Medium to large application

Although the Django includes built-in tools, but its complexity is less suitable for disease prediction system. The Flask is better suited for this project because it is simpler and more flexible, allowing for a quick setup of machine learning prediction APIs. In addition, Flask is suitable for small-scale or prototype applications. Flask also offers the ability to control the flow and design of the application.

2.6.2 Frontend Framework

The frontend is responsible for user engagement with the system. It should be responsive and dynamic in order to attract user's attention and enhance user experience. There are two fronted frameworks need to be considered in this part, which are the React.js and Angular.

2.6.2.1 React.js

The React was built by Facebook in 2013 (GeeksforGeeks, 2023). React.js is a JavaScript library used to designing the user interfaces (UI) for web and mobile application, especially the single page application (SPAs) and allows developers to develop reusable UI elements (W3Schools, 2020). Due to the use of JSX and state management, the learning curve for React is not high. React uses a virtual DOM to minimize the direct DOM updates and ensure the fast rending of dynamic content.

Strengths of React:

- Strong community support. The ecosystem of React provides a wealth of resources and libraries that facilitate the development process.
- Better performance and interactivity. The virtual DOM and reactivity of React ensure a fast, responsive user interface that able to enhance the user's experience in the application.

Weakness of React:

- Learning curve. The JSX and the state management concepts is challenging for the developers and potentially slow down the initial development.

- Additional libraries are needed. This is because React requires additional routing and statement which can add complexity for small application.

2.6.2.2 Angular

Angular is a well-established JavaScript framework, developed using TypeScript, which provides a number of built-in tools for routing and form validation. Angular is commonly used to build robust single-page application (GeeksforGeeks, 2023). The Angular provides bidirectional data binding, and dependency injection. Angular requires setting up the modules and services for building the user interface, which can slow down initial development. Angular is suitable for large-scale applications. In addition, Angular provides development tools to handle and develop the complex applications faster.

Strengths of Angular:

- Scalability. The Angular highly scalable, including built-in tools for large applications.
- Better user experience. The two-way data binding of Angular can synchronize the inputs and outputs automatically. In addition, the dependencies between components are managed by an integrated dependency injection.

Weakness of Angular:

- Learning curve. The learning curve of Angular is steep. The complexity of Angular is challenging for the beginners and can delay the initial development.
- Overkill for small-scale applications. Angular is intended for large projects, where the heavier framework and real-time DOM updates can impact web application load times and performance.

2.6.2.3 Summary of Frontend Framework

Table 2.7: Comparison between React and Angular

Features	React	Angular
Interactivity	High, dynamic UI improves engagement	High, two-way binding
Complexity	Moderate	High
Scalability	High	Very high
Performance	High, virtual DOM for fast updates	Good, real DOM with change detection
Community Support	Excellent, large ecosystem	Strong, smaller ecosystem

Although the Angular is a robust frontend framework, but its complexity and the steeper learning curve can slow down the development process, particularly for the small project. The React is more suitable in this project. The React provides an efficiency and flexible framework. The virtual DOM of React helps in maintaining the excellent performance of the system. React also allow for integration with APIs.

2.7 Summary

In summary, this chapter provides a foundation for the development of a disease prediction web application using machine learning. By reviewing these research papers, it offers different perspectives on the project. Different models have different characteristics and unique advantages and disadvantages. For this project, the most appropriate model is Random Forest. This is because the RT does not the limitations of independent assumptions constraints and can handle noisy data and high dimensional data well.

In addition, this chapter also analyses and compares the strengths and weaknesses of existing web applications. Each of the existing web applications has its own features and strengths. These strengths of the existing web application are utilizing as the project requirements of this project. Furthermore, the evaluation metrics are also important to indicate whether the application is accurate and reliable. In this project, the accuracy, sensitivity (recall), precision and F1 score is selected as the evaluation metrics.

Moreover, the selected data source for this project is the Symptom-Disease Prediction Dataset (SDPD). This is because it supports symptom-disease mapping using binary features, covers 41 diseases, and requires minimal preprocessing. Compared to different frontend and backend frameworks, the preferred web application frameworks are Flask and React. Flask is simpler and more flexible, allowing for quick setup of machine learning prediction APIs. Besides that, the virtual DOM of React helps keep the system's performance excellent.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

The Disease Prediction Web Application using Machine Learning is allow users to input their symptoms and obtain accurate predictions result with a user-friendly interface. This chapter discusses the SDLC methodology used for this project and provides a clear work plan. This chapter presents an outline of system development including integration machine learning model with application. In addition, the tools and technologies to support the development process of the system are defined within this chapter. Furthermore, the project schedule with Work Breakdown Structure (WBS) and Gantt chart to indicates the project timeline. The WBS and Gantt chart can track the progress and made adjustments immediately. By outlining the development process, methodology, and requirements of the project, the project can be clear and well-structured.

3.2 Software Development Life Cycle (SDLC) Methodology

The Software Development Life Cycle Methodology outlines a process of detailed plan, design, develop and testing by developers. SDLC has 7 stages, involving detailed planning, analysis, design, implementation, testing, deployment, maintenance and support (Hossain, 2023). There are many different SDLC methodologies, each characterized by different strengths and weaknesses. Choosing a suitable SDLC methodology is important to ensure the system is developed on time and fulfills the user requirements. This section analyses and compares three approaches which are Waterfall, Spiral, Iterative and Agile.

3.2.1 Waterfall

The Waterfall model is a traditional model that provides a sequential and linear approach (Saravanan *et al.*, 2020). Waterfall is suitable for projects that have clearly defined requirements and without any changes. The waterfall methodology requires that each stage be finalized before moving on to the subsequent. The flow of progress in the waterfall method is unidirectional, with no overlap between each stage. Figure 3.1 shows the Waterfall model process.

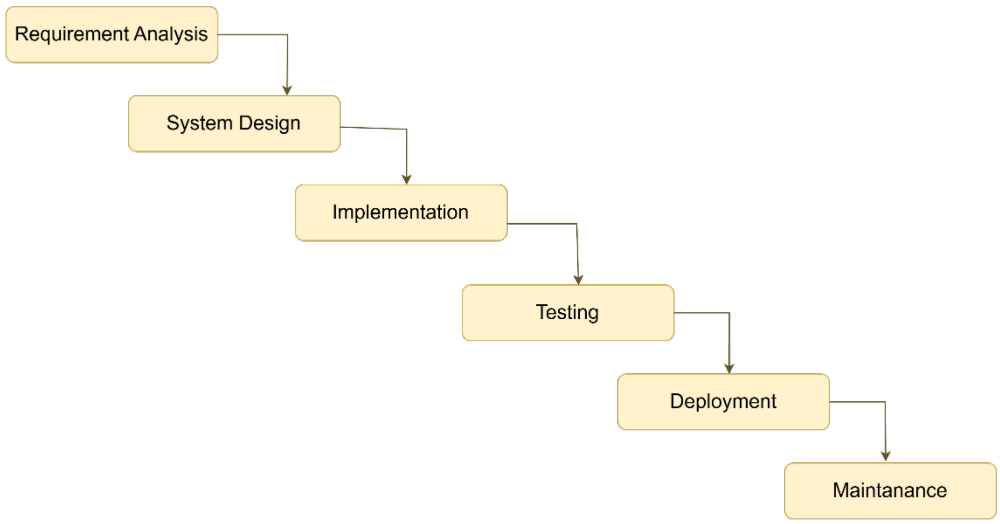


Figure 3.1: Waterfall Model

Advantages of Waterfall:

- Simple to manage and predictable. This makes it easy to track the project progress because it is a one-way model.
- Requirements are well documented and defined. The clear requirements make the project's progress smooth and minimize the changes.
- Clear and structured. The clear structure of Waterfall makes it simple to manage and predict the next steps in the process.

Disadvantages of Waterfall:

- Rigid and inflexible. It assumes that all requirements are clear and nothing has changed, so if the changes occur, it takes more time and money.
- Limited adaptive capacity. Since the testing phase takes place after the completion of the development phase, this limits the ability to adapt to changes and causes problems to be detected later.
- Late detection of problems. Dealing with problems after the development phase is complete can be time-consuming.

3.2.2 Spiral

The Spiral model is used for risk management, which combines Iterative development and Waterfall model (Saravanan *et al.*, 2020). The Spiral is processed through the repetitive cycles, each cycle includes planning, risk analysis, development, and evaluation. It allows for repeated refinement through multiple spirals. Figure 3.2 shows the Spiral model with multiple spirals. Each spiral can refine the project and addressing the risks early. The Spiral can identify the potential errors and reduce them in each iteration. This is suitable for large and complex project, but it is costly and time-consuming because it requires repetitive construction.

Figure 3.2: Spiral Model

Advantages of Spiral:

- Spiral Model is excellent for risk analysis and early detection of potential problems. This can reduce the cost of removing defects after the development process.
- Spiral offer a highly flexible to adjust on requirements or scopes based on the feedback. Users are allowed to involve in the project by testing the prototypes and providing feedback in each spiral.

Disadvantages of Spiral:

- May lead to scope creep. The flexibility of the spiral model allows for adjustments to be made in each spiral, but this may expand the project requirements throughout the project process.
- Complex, costly and time-consuming. The multiple phases make it difficult to manage and track the project process. The repeated works can be costly and time-consuming.

3.2.3 Agile

The Agile model combines iterative and incremental methodology that emphasizes simple, collaboration and flexibility (Hossain, 2023). The Agile allows systems to be built quickly with the ability to change the requirements at any stage of the project life cycle. This methodology focuses on user feedback and the adaptability of making changes to requirements and responding quickly to the changes. This methodology allows for the addition of features and functionality to improve the system. There are several different software development processes in the Agile field, which are Extreme Programming (XP), Kanban, Crystal, Scrum and other. Figure 3.3 shows the 6 stages of the Agile model.

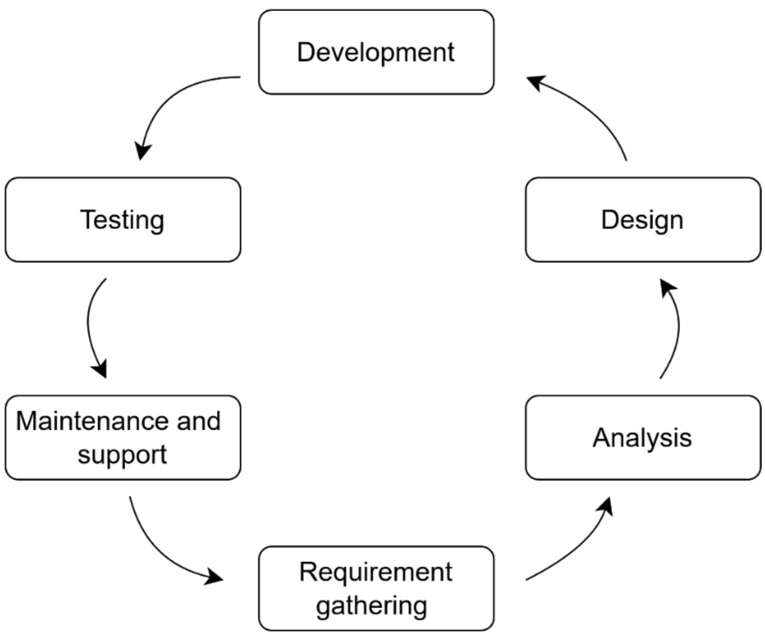


Figure 3.3: Agile Model

Advantages of Agile:

- High flexibility. Agile flexibility adapts to changing needs and is user- centred. New features can be added in Agile even late in the development process.
- High quality and low risk. Agile focuses on frequent testing techniques to minimize the defects and improve system reliability. This helps in identifying and resolving the defects early in the project life cycle.

Disadvantages of Agile:

- Lack of focus on documentation. This can be a challenging task in Agile, which focuses on software development rather than the documentation.
- Predictability is limited. This is difficult to predict deadlines for agile projects because of changing requirements make it difficult to estimated timelines and costs.

3.2.4 Summary of SDLC Methodology

Table 3.1: Comparison of different SDLC Methodology

	Waterfall	**Spiral**	**Agile**
Process Structure	Linear, sequential	Iterative cycles	Iterative sprints
Flexibility	Low	Moderate to High	Very high
User involvement	Low	High, user feedback on prototypes	Very high, continuous feedback in sprints
Requirement	Defined in early stage	Throughout the project period	Throughout the project period
Phases	Sequential phases	Iterative cycles with risk-driven phases	Iterative sprints with

			collaborative phases
Risk Management	Poor	Excellent	Very good
Testing	Testing after development process	Continuous testing throughout the development	Continuous testing throughout development
Complexity	Low	High	Moderate

Each of the methodology has its own advantages and disadvantages, and different project is suitable for different software development methodology. Waterfall is suitable for projects where the requirements are clearly defined and unlikely to change. The Waterfall is less flexible, cannot adapt to changes, and one phase must be completed before progressing to the next. In contrast, the Spiral model is suitable for high-risk projects because Spiral is excellent in risk management. However, it is more complex compared to other methodology. Agile is suitable for projects that required changes throughout the project life cycle because it is flexible and adaptable. In short, the Agile methodology aligns with the project that offering high flexibility and collect user feedback for improvement. Agile allows for quick adaptation to user feedback and the ability to test the system frequently and reduce risks early. In addition, the Agile approach ensures rapid development of web applications with different functionalities. However, the Agile is lack of focus on documentation and unpredictable. This is difficult to estimate the project deadline. Based on this comparison, the Waterfall model was selected for this project due to its clear structure, well-defined requirements, and suitability for academic project timeline.

3.3 Waterfall Methodology

Software Development Life Cycle (SDLC) methodology provides a structured framework for managing the development of disease prediction web applications. In this project, the Waterfall methodology is selected as the SDLC methodology. Waterfall is suitable for projects with the clear requirements and no changes. The requirements for this project are well defined and less variable. In addition, Waterfall is suitable for progress tracking and deliverables. This helps in estimating the project timeline and budget well. Figure 3.4 shows flowchart of the Waterfall methodology for this project. There are 6 phases in the Software Development Flowchart which are Requirements Analysis, System Design, Machine Learning Model development. Web Application development, Integration of Machine Learning Model with Application, and the last is System Testing.

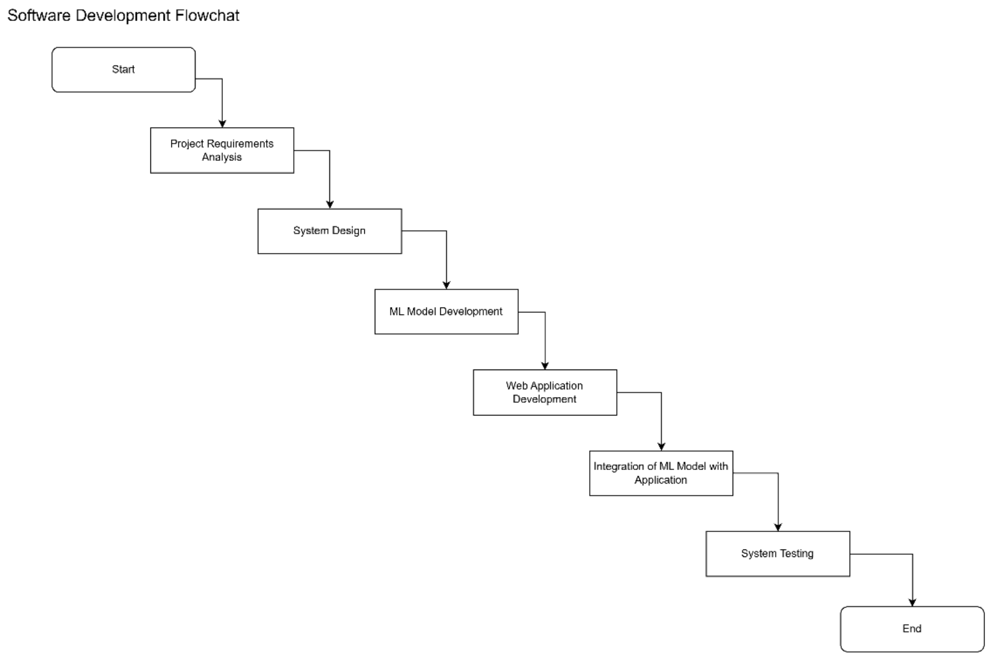


Figure 3.4: Waterfall Methodology Flowchart

3.4 Requirements Analysis

The project requirements can be gathered through several ways. In this project, the requirements were gathered by reviewing existing similar web applications. Interviews and questionnaires are not required for requirements gathering in this project. In chapter 2, there are several similar existing disease prediction web applications have been studied, which are Symptomate, WebMD Symptom Checker, and Your.MD (Healthily). A common feature of these web applications is the ability to allow users to enter symptoms via optional options or types. In this project, the system provides two options for users to enter the symptoms which are a predefined dropdown list of symptoms or manually enter symptoms via a free-text field. This is a feature that has not yet been implemented in current disease prediction web applications. Additionally, the web application can predict the potential disease a user may have and return the results to the user thought a responsive user interface. The results include the potential disease and the recommendations further actions to be taken by the user. Furthermore, the system can store the history of users, such as user inputs, predicted results, and medical advice for the user. This is another feature that other web applications have not yet implemented. The non-functional requirements and functional requirement will be produced at the end of this phase. The detailed functional and non-functional requirements are discussed further in Chapter 4.

3.5 System Design

In Waterfall methodology, the system design phase follows the requirements analysis phase. The system design phase defines the system architecture and the data flow to ensure that the system is fulfil the functional and non-functional requirements. It includes the design of user interface, data flows, backed API architecture and integration with machine learning model. The design also includes the chosen tools and technology in preparation for the implementation phase of the project. Figure 3.5 presents the data flow diagram of the project. The data flow diagram demonstrates the process of processing user symptoms, generating predicted results, and clearly displaying results to meet program requirements. This phase provides a clear insight of the development process to ensure that the performance and usability standards. The deliverable for this phase is the generation of the system architecture design proposal.

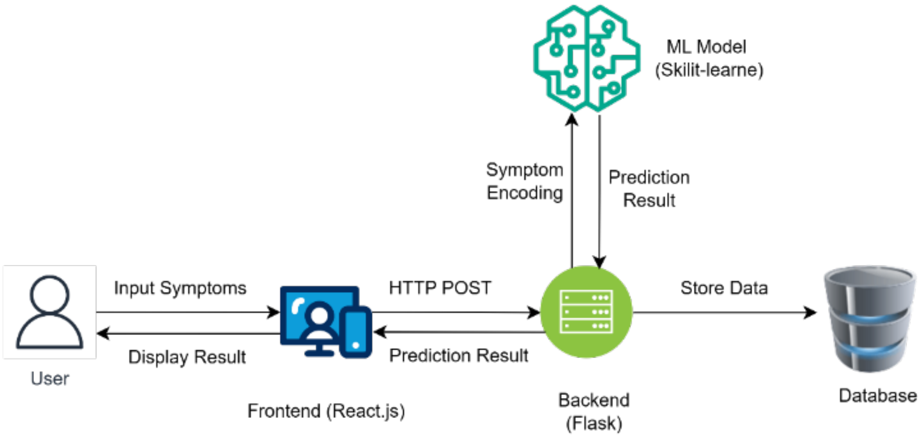


Figure 3.5: System Design Diagram

3.6 Implementation Phase

There are 3 steps perform in the Implementation Phase, which are Machine Learning Model Development, Web Application Development and Integration of ML Model with Web Application. The deliverables of this phase are the completion of the machine learning model development, web application development and the integration of the ML model with web application.

3.6.1 Machine Learning Model Development

The key component of this project is developing the machine learning (ML) model for the Disease Prediction Web Application. This part outlines the 6 phases ML model development process. These 6 phases include model selection, dataset selection, data preprocessing, feature selection, model construction and model evaluation. These phases are important for ensure that the model aligns with the project requirements and user expectations.

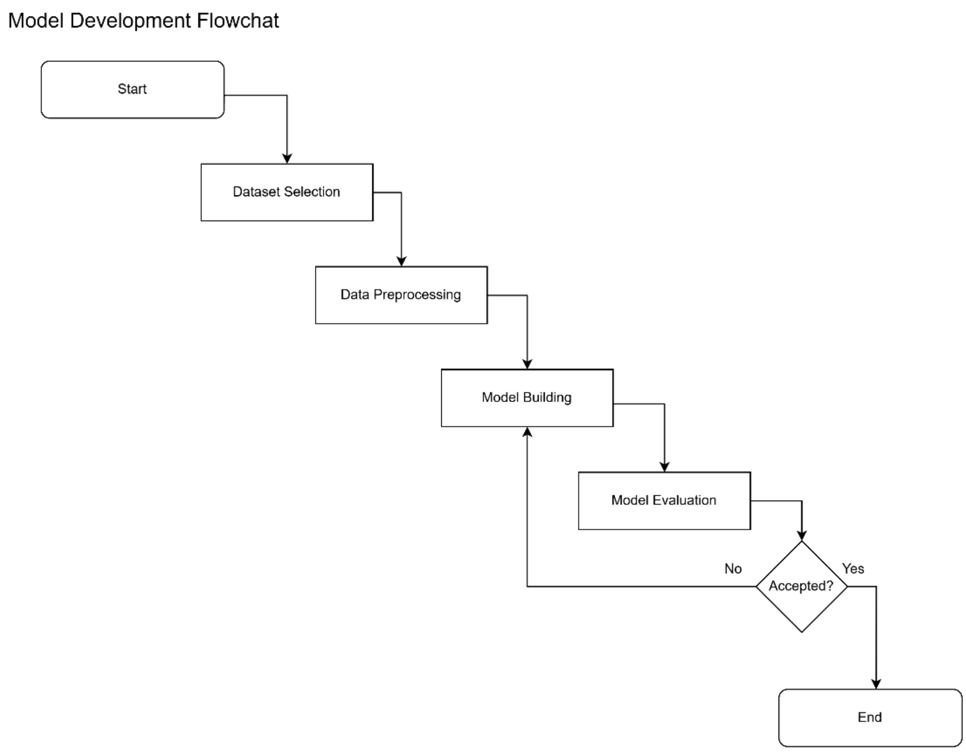


Figure 3.6: Model Development Flowchart

3.6.1.1 Model Selection

Model Selection involves selecting the most appropriate ML algorithm for the classification task of predicting diseases based on symptoms. After reviewing the relevant literature on different models, 3 ML algorithms which are Decision Tree, Random Forest, and Support Vector Machine are selected and assessed based on their performance.

- **Decision Tree (DT)**: The DT is considered because it is interpretability and provides the explicit decision rules. For example, if the cough = yes, predict disease X. This helps explain the prediction results to the user. However, it may overfit the complex datasets.

- **Random Forest (RT)**: The RT is considered because its robustness and high accuracy. RT is combined with multiple DT to reduce the overfit. It also can handle the missing data well.

- **Support Vector Machine (SVM)**: The SVM is considered because its capability to handle high-dimensional data. SVM is effective in binary classification with clear margins. However, it is not suitable for large datasets because of the long computation time.

The selection of these models is based on their complementary advantages to allow for a comparative evaluation to determine the most appropriate model for the project.

3.6.1.2 Dataset Selection

The dataset selection phase is to determine the data source for the training and evaluating the ML models. The selected dataset is the Symptom-Disease Prediction Dataset (SDPD). This dataset was also identified in Chapter 2, and the SDPD was the most suitable dataset for this project compared to the different other datasets. This dataset is available on Mendeley Data. The SDPD dataset contains 4,920 instances and 132 symptoms feature covering 41 unique diseases. The structure of the dataset is a table with multiple rows for the patient cases and multiple columns for the symptoms and disease labels. The symptoms are binary values, with 1 indicating presence and 0 indicating absence. The dataset is relevant to this project as it supports a wide range of diseases and can fulfil the functional requirements of the project.

3.6.1.3 Data Preprocessing

Data preprocessing is an important stage in preparing the SDPD dataset for model training and testing. This process aims to resolve the quality issues and convert the raw data into a clean and structured format. There are several steps in this phase to ensure that the ML model performs well and effectively. These steps can be implemented by using Scikit-learn and Python libraries.

Steps involved:

- **Handling missing data**. This helps to impute missing values in the SDPD dataset and prevent model errors. For binary symptom features, the missing values are replaced with the most frequently occurring values (mode), as this maintains the distribution of the dataset.

- **Categorical Label Encoding**. The SDPD dataset contain binary symptom features, but the dataset includes the categorical disease labels such as “Fungal Infection” rather than numerical indexes. Therefore, coding using Scikit-learn's LabelEncoder is required to convert the categorical disease labels to integers.
- **Duplicate Removal.** Duplicate records may introduce bias into the training process and potentially reduce processing speed, particularly when handling relatively small datasets. To address this issue, the dataset underwent duplicate scanning, and all duplicate rows were removed.
- **Data Splitting.** By using Scikit-learn, the dataset can divide into three subsets, which are training set, validation set and testing set. The dataset can be split into 70-15-15 parts, which indicates that 70% for training, 15% for validating and 15% for testing.

These steps ensure that the data required for model training is clean and reliable and reduce potential risks such as low model performance due to data quality issues.

3.6.1.4 Model Building

This phase consists of training selected ML models including support vector machine, random forest, and decision tree on pre-processed SDPD datasets. By using Scikit-learn to implement machine learning models and ensure the models are available for evaluation. Firstly, each model is initialized using the default parameter such as ‘DecisionTreeClasssifier()’, ‘RandomForestClassifier(n_estimators=100)’ and ‘SVC(probability=True)’. Furthermore, it is required for hyperparameter tuning. The GridSearchCV or RandomizedSearchCV are used to fine-tuning the hyperparameters to improved performance. In addition, the next step is model fitting. The models is trained by using training set, such as X_train and y_train. For instance, by using the model.fit(X_train, y_train) to train the models. This phase helps to reduce the risks of overfitting.

3.6.1.5 Model Evaluation

Model evaluation is the step of evaluates the performance of the trained models on test sets and select the most suitable model for project deployment. For this disease prediction web application, various evaluation metrics were used to comprehensively analyse model performance, especially for the classification task. The key evaluation metrics include sensitivity and accuracy. Sensitivity can minimize the false negatives results such as missed diagnoses. Accuracy is frequently used to measure the overall correctness of machine learning models. The evaluation compares all three models to select the best performer.

3.6.2 Web Applications Development

The Web Applications Development phase aims to building the core components of the Disease Prediction Web Application using Machine Learning. This ensures that a functional and user-friendly application is developed before integrating it with the machine learning model. The development process includes building the backend using Flask to handle API requests, creating the frontend with React.js to provide an interactive user interface, and make sure that the components meet functional and non-functional requirements. The backend is responsible for handling HTTP requests, processing user inputs, and preparing the application for ML model integration. During this phase, the Flask application sets up with the necessary routes and configurations to support the functionality of the application. In addition, the frontend is responsible for providing a responsive and user-friendly interface for users to input symptoms and view prediction results, fulfilling the functional requirements and a clear output display. During this phase, the React.js application is set up the necessary components and routes to support user interactions.

3.6.3 Integration of ML Model with Web Application

The integration of the ML model with web application is important in this project. This ensures that the well-trained model is able to process user input symptoms and deliver the prediction results through the web application's interface. This process combines the ML model development with the web application development. The integration included loading and saving the ML model using scikit-learn, creating the API endpoint using Flask to provide prediction service, and enabling the frontend to send the symptoms and display the prediction results by using React. This phase can ensure that the ML model can communicate well with the backend to provide the real-time predictions and responses based on user input. The integration process ensures that the robust and efficient system is developed and aligned with the requirements. Furthermore, the Google Gemini will also integrate with the web application to generate general medical advice for each potential disease. The completed system will be produced at the end of the phase.

3.7 System Testing

System testing is the testing phase in the Waterfall methodology. It is important for the reliability of Disease Prediction Web Application using Machine Learning. This phase ensures that the integrated system is fulfil the project requirements. The objective of testing is to validate that the web application achieves at least 85% sensitivity for disease prediction and able to predict the diseases accurately.

The system testing also includes testing functional requirements and non-functional requirements. The testing phase can mitigate the potential risks such as error predictions by evaluating the performance and reliability of the system. There are various types of testing can be performed within the project, including unit testing, integration testing, usability testing and user acceptance testing. These testing helps ensure that that software system fulfils the specific requirements and reduce potential risks such as inaccurate prediction of results and incorrect medical advice. In this phase, the test report and bug list will be produced at the end.

3.8 Tools and Technologies

These tools and technologies are important for development of the Disease Prediction Web Application to ensure the efficient and effective implementation, testing, and deployment. This section describes the details of the tools used in this project. For this project, the selected tools include Flask for the backend, React for the frontend, Scikit-learn for the model framework, MySQL for the database, GitHub for task tracking and Postman for testing.

3.8.1 Flask

Flask is selected as the backend framework for this project. Flask is responsible for handle the server-side request. Flask is characterized by lightweight and rapid development. Flask can ensure the fast performance and quick setup due to its small footprint, making it suitable for small to medium-sized projects. In addition, Flask is simple, allowing developers to customize the structure of the application to their requirements and to build APIs and integrate machine learning models easily and quickly.

3.8.2 React

React is selected as the frontend framework for this project. React is responsible for build the interactive and user-friendly interface. React has strong community support. The ecosystem of React provides a wealth of resources and libraries that facilitate the development process. Moreover, the React offer better performance and interactivity than other frontend frameworks. The virtual DOM and reactivity of React ensure a fast, responsive user interface that enhance the user’s experience in the application.

3.8.3 Scikit-learn

Scikit-learn is selected as the machine learning framework for this project. Scikit-learn is a Python library that provide an efficient and simple API for traditional machine learning algorithms to predictive data analysis. Due to its simplicity, comprehensiveness and consistency, it is widely utilizing in the field of data science and machine learning. Scikit-learn also provides data preprocessing, feature selection, evaluation tools to ensure that the ML model meet the requirements.

3.8.4 MySQL

In this project, MySQL is chosen as the database management system. The MySQL is responsible for storing the history of prediction such as user inputs, prediction results, and medical recommendations. MySQL is widely used for managing and storing the structured data. The SQL means Structured Query Language that support data retrieval and manipulation. MySQL is reliable and can integrates well with Flask, allowing the backend to store user data such as input symptoms and prediction results.

3.8.5 GitHub

In this project, GitHub is chosen as the version control tool to ensure the efficient code management. GitHub is a cloud-based platform with a repository that allow developers to perform version control such as commits, pull requests and more. This is important for tracking changes to the codebase. GitHub also allow developers to create different branches to develop new features without affecting the main codebase. In addition, it allows for backup and restoration of data, thus preventing data loss.

3.8.6 Postman

In this project, the Postman is chosen as the API testing tool. Postman is responsible for sending HTTP requests and return the prediction responses to the users. It is important for testing and managing the application interface. Postman provides a user-friendly interface and able to perform features without additional code. Postman can also verify that the actual results match the expected results to ensure that the accurate results are returned.

3.9 Project Plan

The project plan for the Disease Prediction Web Application using Machine Learning provides a structured approach to managing the development process, outlining the timeline and deliverables to ensure the well-structured development process.

3.9.1 Work Breakdown Structure (WBS)

0.0 Disease Prediction Web Application using Machine Learning

1.0 Project Preparation
- 1.1 Study the interested proposal title
- 1.2 Discuss with supervisor
- 1.3 Confirm FYP title

2.0 Project Initiation
- 2.1 Draft Chapter 1: Introduction
  - 2.1.1 General Introduction
  - 2.1.2 Define Important of study
  - 2.1.3 Define Problem Statements
  - 2.1.4 Define Aim and Objectives
  - 2.1.5 Define Scope and Limitations
  - 2.1.6 Propose Project Solution

3.0 Literature Review
- 3.1 Draft Chapter 2: Literature Review
  - 3.1.1 Research and Compare Models
  - 3.1.2 Compare Existing Web Apps
  - 3.1.3 Define Evaluation Metrics
  - 3.1.4 Identify Dataset Sources
  - 3.1.5 Compare Web Application Framework
- 3.2 Review and Finalize Chapter 1
  - 3.2.1 Review draft Chapter 1 with supervisor
  - 3.2.2 Incorporate feedback and finalize

4.0 Methodology and Work Plan
- 4.1 Draft Chapter 3: Methodology and Work Plan
  - 4.1.1 Compare SDLC Methodologies
  - 4.1.2 Discuss Tools and Technologies
  - 4.1.3 Create WBS
  - 4.1.4 Create Gantt Chart

4.2 Review and Finalize Chapter 2
4.2.1 Review draft Chapter 1 with supervisor
4.2.2 Incorporate feedback and finalize
5.0 Requirements Analysis
5.1 Functional Requirements
5.2 Non-Functional Requirements
5.3 Develop use case diagram and description
5.4 Review and Finalize Chapter 3
5.5 Develop Prototype
5.5.1 Develop Low Fidelity Prototype
6.0 System Design
6.1 Define system architecture and the data flow
7.0 System Development
7.1 Machine Learning Model development
7.1.1 Model Selection
7.1.2 Dataset Selection
7.1.3 Data preprocessing
7.1.4 Model Building
7.1.5 Model Evaluation
7.2 Web Application Development
7.2.1 Develop Fronted
7.2.2 Develop Backend
7.3 Integration of ML Model with Web App
7.3.1 Load ML model into Flask
7.3.2 Integrate the LLM model into existing application
8.0 System Testing
8.1 Unit Testing
8.2 Integration Testing
8.3 User acceptance Testing
8.4 User Interface Design Feedback
9.0 Closing
9.1 Finalize project documentation
9.2 Submit Project

3.9.2 Gantt Chart

3.9.2.1 Overview of the Disease Prediction Web Application using Machine Learning Timeline

Figure 3.7 shows an overview of the project timeline. The Gantt chart below illustrates the tasks required to complete the project. These tasks include project preparatory, project initiation, literature review, methodology and work plan, requirement analysis, system design, system development, system testing, and closing. The estimated total duration to complete the project is 219 days.

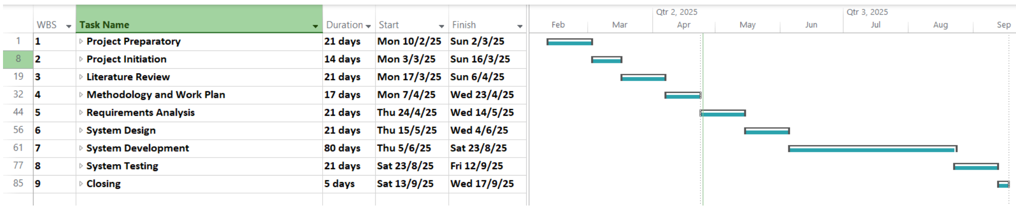


Figure 3.7: Overview of the Project Timeline

3.9.2.2 Project Preparatory and Project Initiation Timelines

Figure 3.8 shows the tasks included in project preparatory and project initiation timelines. The estimated duration of project preparatory is 21 days and project initiation are 14 days.

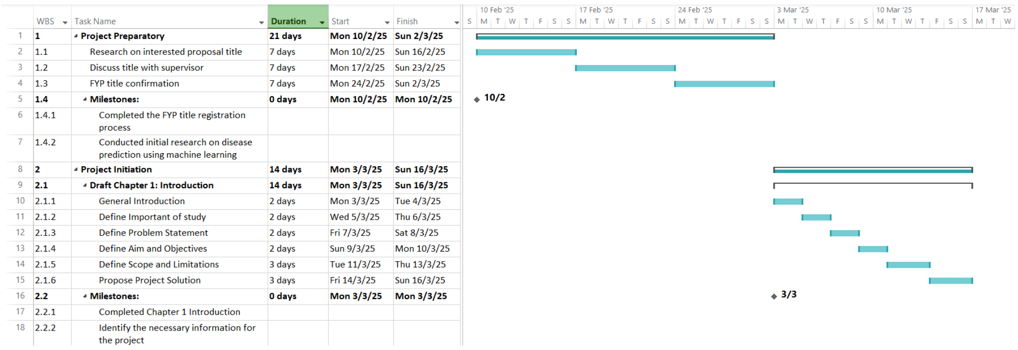
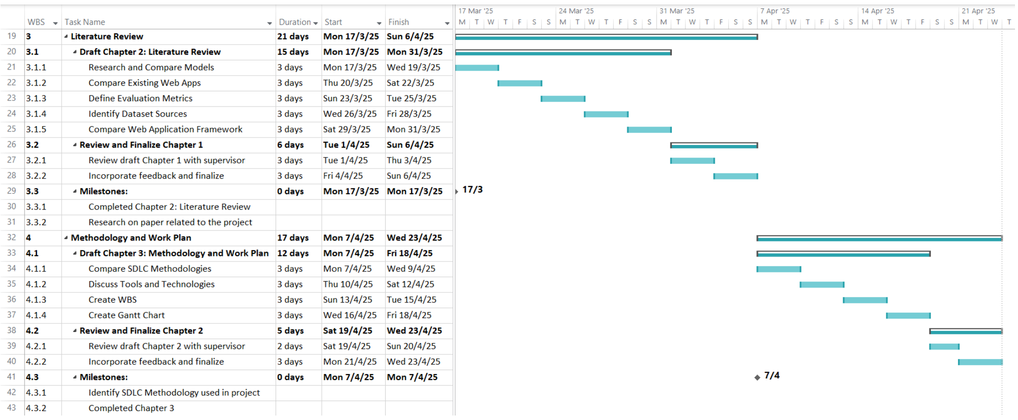


Figure 3.8: Project Preparatory and Project Initiation Timelines

3.9.2.3 Literature Review and Methodology Timelines

Figure 3.9 shows the timelines for the literature review and methodology phases. These tasks include drafting Chapter 2, reviewing and finalizing Chapter 1, drafting Chapter 3 under the methodology timelines. The estimated duration for the literature review is 21 days and for the methodology and work plan is 17 days.

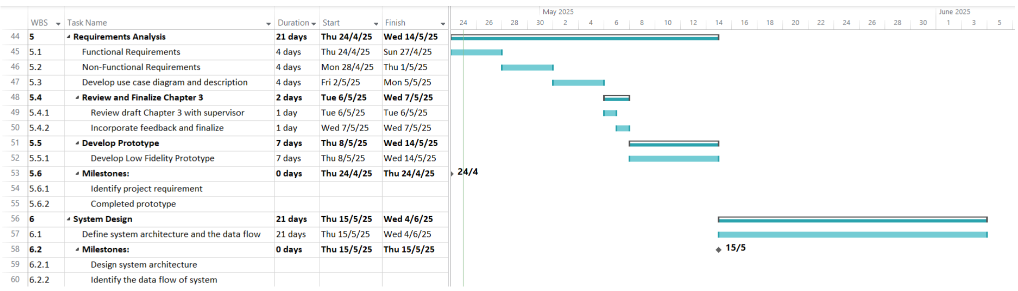


	WBS	Task Name	Duration	Start	Finish
19	**3**	**Literature Review**	**21 days**	**Mon 17/3/25**	**Sun 6/4/25**
20	**3.1**	**Draft Chapter 2: Literature Review**	**15 days**	**Mon 17/3/25**	**Mon 31/3/25**
21	3.1.1	Research and Compare Models	3 days	Mon 17/3/25	Wed 19/3/25
22	3.1.2	Compare Existing Web Apps	3 days	Thu 20/3/25	Sat 22/3/25
23	3.1.3	Define Evaluation Metrics	3 days	Sun 23/3/25	Tue 25/3/25
24	3.1.4	Identify Dataset Sources	3 days	Wed 26/3/25	Fri 28/3/25
25	3.1.5	Compare Web Application Framework	3 days	Sat 29/3/25	Mon 31/3/25
26	**3.2**	**Review and Finalize Chapter 1**	**6 days**	**Tue 1/4/25**	**Sun 6/4/25**
27	3.2.1	Review draft Chapter 1 with supervisor	3 days	Tue 1/4/25	Thu 3/4/25
28	3.2.2	Incorporate feedback and finalize	3 days	Fri 4/4/25	Sun 6/4/25
29	**3.3**	**Milestones:**	**0 days**	**Mon 17/3/25**	**Mon 17/3/25**
30	3.3.1	Completed Chapter 2: Literature Review			
31	3.3.2	Research on paper related to the project			
32	**4**	**Methodology and Work Plan**	**17 days**	**Mon 7/4/25**	**Wed 23/4/25**
33	**4.1**	**Draft Chapter 3: Methodology and Work Plan**	**12 days**	**Mon 7/4/25**	**Fri 18/4/25**
34	4.1.1	Compare SDLC Methodologies	3 days	Mon 7/4/25	Wed 9/4/25
35	4.1.2	Discuss Tools and Technologies	3 days	Thu 10/4/25	Sat 12/4/25
36	4.1.3	Create WBS	3 days	Sun 13/4/25	Tue 15/4/25
37	4.1.4	Create Gantt Chart	3 days	Wed 16/4/25	Fri 18/4/25
38	**4.2**	**Review and Finalize Chapter 2**	**5 days**	**Sat 19/4/25**	**Wed 23/4/25**
39	4.2.1	Review draft Chapter 2 with supervisor	2 days	Sat 19/4/25	Sun 20/4/25
40	4.2.2	Incorporate feedback and finalize	3 days	Mon 21/4/25	Wed 23/4/25
41	**4.3**	**Milestones:**	**0 days**	**Mon 7/4/25**	**Mon 7/4/25**
42	4.3.1	Identify SDLC Methodology used in project			
43	4.3.2	Completed Chapter 3			

Figure 3.9: Literature Review and Methodology Timelines

3.9.2.4 Requirements Analysis and System Design Timelines

Figure 3.10 shows the timelines for requirements analysis and system design phases. The milestones for requirements analysis phase include identifying the project requirements and completing the prototype. The milestones for system design phase include designing the system architecture and identifying the data flow of the system. The estimated duration for both the requirements analysis and system design phases is 21 days each.

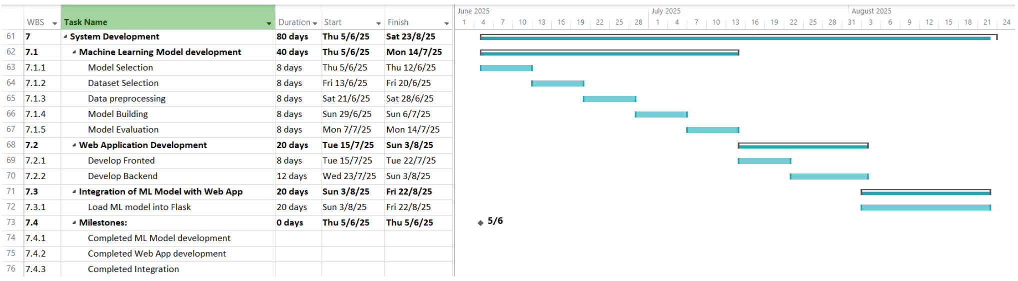


	WBS	Task Name	Duration	Start	Finish
44	**5**	**Requirements Analysis**	**21 days**	**Thu 24/4/25**	**Wed 14/5/25**
45	5.1	Functional Requirements	4 days	Thu 24/4/25	Sun 27/4/25
46	5.2	Non-Functional Requirements	4 days	Mon 28/4/25	Thu 1/5/25
47	5.3	Develop use case diagram and description	4 days	Fri 2/5/25	Mon 5/5/25
48	**5.4**	**Review and Finalize Chapter 3**	**2 days**	**Tue 6/5/25**	**Wed 7/5/25**
49	5.4.1	Review draft Chapter 3 with supervisor	1 day	Tue 6/5/25	Tue 6/5/25
50	5.4.2	Incorporate feedback and finalize	1 day	Wed 7/5/25	Wed 7/5/25
51	**5.5**	**Develop Prototype**	**7 days**	**Thu 8/5/25**	**Wed 14/5/25**
52	5.5.1	Develop Low Fidelity Prototype	7 days	Thu 8/5/25	Wed 14/5/25
53	**5.6**	**Milestones:**	**0 days**	**Thu 24/4/25**	**Thu 24/4/25**
54	5.6.1	Identify project requirement			
55	5.6.2	Completed prototype			
56	**6**	**System Design**	**21 days**	**Thu 15/5/25**	**Wed 4/6/25**
57	6.1	Define system architecture and the data flow	21 days	Thu 15/5/25	Wed 4/6/25
58	**6.2**	**Milestones:**	**0 days**	**Thu 15/5/25**	**Thu 15/5/25**
59	6.2.1	Design system architecture			
60	6.2.2	Identify the data flow of system			

Figure 3.10: Requirements Analysis and System Design Timelines

3.9.2.5 System Development Timelines

Figure 3.11 shows the system development timelines. The estimated duration this phase is 80 days. The tasks include machine learning model development, web application development and integration of machine learning model with the web application. The milestones for this phase include completing the ML model and web application development and finalizing the integration.

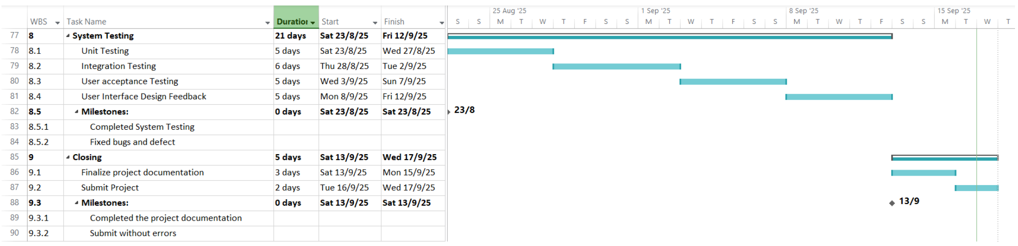


	WBS	Task Name	Duration	Start	Finish
61	**7**	**System Development**	**80 days**	**Thu 5/6/25**	**Sat 23/8/25**
62	**7.1**	**Machine Learning Model development**	**40 days**	**Thu 5/6/25**	**Mon 14/7/25**
63	7.1.1	Model Selection	8 days	Thu 5/6/25	Thu 12/6/25
64	7.1.2	Dataset Selection	8 days	Fri 13/6/25	Fri 20/6/25
65	7.1.3	Data preprocessing	8 days	Sat 21/6/25	Sat 28/6/25
66	7.1.4	Model Building	8 days	Sun 29/6/25	Sun 6/7/25
67	7.1.5	Model Evaluation	8 days	Mon 7/7/25	Mon 14/7/25
68	**7.2**	**Web Application Development**	**20 days**	**Tue 15/7/25**	**Sun 3/8/25**
69	7.2.1	Develop Fronted	8 days	Tue 15/7/25	Tue 22/7/25
70	7.2.2	Develop Backend	12 days	Wed 23/7/25	Sun 3/8/25
71	**7.3**	**Integration of ML Model with Web App**	**20 days**	**Sun 3/8/25**	**Fri 22/8/25**
72	7.3.1	Load ML model into Flask	20 days	Sun 3/8/25	Fri 22/8/25
73	**7.4**	**Milestones:**	**0 days**	**Thu 5/6/25**	**Thu 5/6/25**
74	7.4.1	Completed ML Model development			
75	7.4.2	Completed Web App development			
76	7.4.3	Completed Integration			

Figure 3.11: System Development Timelines

3.9.2.6 System Testing and Closing Timelines

Figure 3.12 shows the timelines for system testing and closing phases. The estimated duration for system testing phase is 21 days. The tasks in this phase include unit testing, integration testing, user acceptance testing and user interface design feedback. The closing phase is estimated to takes 5 days and involves completing the project documentation.



	WBS	Task Name	Duration	Start	Finish
77	**8**	**System Testing**	**21 days**	**Sat 23/8/25**	**Fri 12/9/25**
78	8.1	Unit Testing	5 days	Sat 23/8/25	Wed 27/8/25
79	8.2	Integration Testing	6 days	Thu 28/8/25	Tue 2/9/25
80	8.3	User acceptance Testing	5 days	Wed 3/9/25	Sun 7/9/25
81	8.4	User Interface Design Feedback	5 days	Mon 8/9/25	Fri 12/9/25
82	**8.5**	**Milestones:**	**0 days**	**Sat 23/8/25**	**Sat 23/8/25**
83	8.5.1	Completed System Testing			
84	8.5.2	Fixed bugs and defect			
85	**9**	**Closing**	**5 days**	**Sat 13/9/25**	**Wed 17/9/25**
86	9.1	Finalize project documentation	3 days	Sat 13/9/25	Mon 15/9/25
87	9.2	Submit Project	2 days	Tue 16/9/25	Wed 17/9/25
88	**9.3**	**Milestones:**	**0 days**	**Sat 13/9/25**	**Sat 13/9/25**
89	9.3.1	Completed the project documentation			
90	9.3.2	Submit without errors			

Figure 3.12: System Testing and Closing Timelines

CHAPTER 4

PROJECT SPECIFICATIONS

4.1 Introduction

This chapter details the specifications for Disease Prediction Web Application using Machine Learning, providing the details information on the system requirements, design and functionality. This section presents the functional and non-functional requirements specifications, use case modelling with use case diagram and use case description, proposed system flow, interface system flow and low-fidelity prototype. Use case diagram demonstrates the interaction between user and system. This chapter ensures that the system is aligned with the goals and objectives of this system.

4.2 Requirements Specification

This section describes the functional and non-functional requirements of the project to ensure that the system fulfil the user expectations and requirements. The requirements are gathered from existing similar disease prediction web applications, which combine the strengths of the existing systems.

4.2.1 Functional Requirements Specification

The functional requirements define the specific functions of the system that provide to fulfil the user’s needs. The Table 4.1 identifies the functional requirements for the Disease Prediction Web Application using Machine Learning

Table 4.1: Functional Requirements

ID	Functional Requirement Statements
FR001	The system shall allow user to register an account.
FR002	The system shall allow user to login with the email and password.
FR003	The system shall allow user to input symptoms using predefined checklist or free text.

FR004	The system shall display the prediction results to user with explanation.
FR005	The system shall provide medical advice to the user based on the predicted results.
FR006	The system shall allow user to store the predicted results into the database.
FR007	The system shall allow user to view the historical symptoms and prediction results.
FR008	The system shall allow user to view their profile.
FR009	The system shall allow user to update their personal data such as username, date of birth, gender and password.

4.2.2 Non-functional Requirements Specification

The non-functional requirements specify the performance, usability and reliability of the system. This can ensure that the system fulfil the quality standards. The Table 4.2 identifies the non-functional requirements for the project.

Table 4.2: Non-functional requirements

ID	**Non-Functional Requirements Statements**
NFR001	The system shall provide a responsive and user-friendly interface and clear navigation.
NFR002	The system shall be secured and able to protect the predicted results and symptoms entered by the user.
NFR003	The response time of the system should be responsive when the user submits the symptoms.
NFR004	The system should be available all the time.
NFR005	The system should be compatible with popular browsers including Google Chrome and Microsoft Edge.

4.3 Use Case Modelling

The use case modelling describes the interaction between the user and the system. This clearly illustrates the uses of web applications. This section includes use case diagram and use case descriptions.

4.3.1 Use Case Diagram

The Figure 4.1 shows the use case diagram for disease prediction web application using machine learning system.

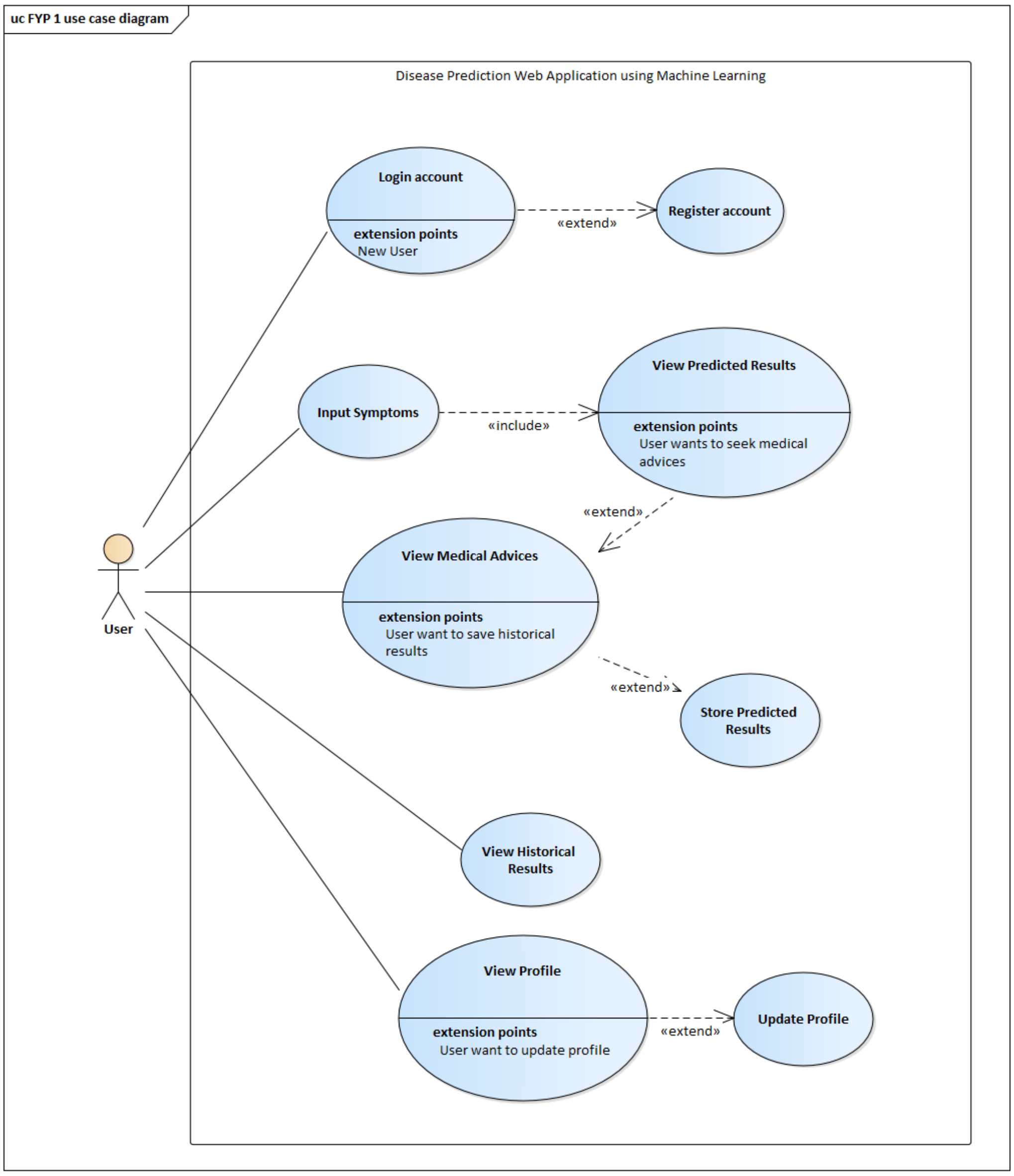


Figure 4.1: Use case diagram

4.3.2 Use Case Description

4.3.2.1 Login account

Table 4.3: Use case description of Login Account

<table>
<tr><td colspan="2">Use Case Name: Login account</td><td>ID:
UC001</td><td>Importance Level:
High</td></tr>
<tr><td>Primary Actor: User</td><td colspan="3">Use Case Type: Details, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
User – wants to login to the account to access the web application by using email and password.</td></tr>
<tr><td colspan="4">Brief Description: This use case describes how a user login to the account to access features of the web application.</td></tr>
<tr><td colspan="4">Trigger: The user wants to access the disease prediction web application.</td></tr>
<tr><td colspan="4">Relationships:
Association : User
Include : -
Extend : Register account
Generalization: -</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The system displays 2 options for using the system. Perform 1.1 or 1.2.
1.1 If the store owner selects the “Login” option, the <u>flow no.2</u> continues.
1.2 If the store owner selects the “Register” option, the flow ends.
2. The system displays the login screen.
3. The user enters the email and password.
4. The system checks and validates the credentials. Perform 4.1 or 4.2.
4.1 If the email and password are valid, flow no.5 continues.
4.2 If the email and password are invalid, the system will indicate that the login was unsuccessful, the flow no.1 continues.
5. The user successfully logged into the web application and can access the functions in the system.</td></tr>
</table>

Sub-flows**: -**
Alternate/Exceptional Flows: 1a. If the user selects "Register," the system executes the Register Account use case.

4.3.2.2 Register account

Table 4.4: Use case description of Register account

<table>
<tr><td colspan="2">Use Case Name: Register account</td><td>ID:
UC002</td><td>Importance Level:
High</td></tr>
<tr><td>Primary Actor: User</td><td colspan="3">Use Case Type: Details, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
User – wants to register an account to access the features of disease prediction web application.</td></tr>
<tr><td colspan="4">Brief Description: This use case describes how a user can register an account to gain access to the web application.</td></tr>
<tr><td colspan="4">Trigger: The user wants to become a user.</td></tr>
<tr><td colspan="4">Relationships:
Association : User
Include : -
Extend : -
Generalization: -</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The system displays 2 options for using the system. Perform 1.1 or 1.2.
1.1 If the user selects the "Register" option, the flow no.2 continues.
1.2 If the user selects the "Login" option, the flow ends.
2. The system displays the registration form.
3. The user logs in with username and password.
4. The system validates the information provided. Perform 4.1 or 4.2.
4.1 If the information is valid, flow no.5 continues.
4.2 If the information is invalid, flow no.6 continues.</td></tr>
</table>

5. The system indicates successful registration. 6. The system indicates unsuccessful registration.
Sub-flows**:**
Alternate/Exceptional Flows: 1a. If the user selects "Login," the system proceeds to the **Login** use case.

4.3.2.3 Input Symptoms

Table 4.5: Use case description of Input Symptoms

Use Case Name**: Input Symptoms**		ID**:** **UC003**	Importance Level**:** **High**
Primary Actor**: User**	Use Case Type**: Details, Essential**		
Stakeholders and Interests**: User – wants to input symptoms via the checklist.**			
Brief Description**: This use case describes how a user enters the symptoms into the system for prediction of diseases using a dropdown menu or free text.**			
Trigger**: The user wants to input their symptoms to predict potential diseases.**			
Relationships: Association **: User** Include **: -** Extend **: View Predicted Results** Generalization**: -**			
Normal Flow of Events**:** 1. The system displays two input options which are dropdown checklist by listing 132 symptoms and a free text input box. *Perform 2.1 or 2.2* 2.1. If the user selects input symptoms by using dropdown menu, the **flow no.3** continues. 2.2 If the user selects input symptoms via free text, the **flow no.4** continues.			

2. The user selects one or more symptoms by checking the corresponding boxes. 3. The user enters the symptoms in the text input box. 4. The system displays a preview or list of the selected symptoms to the user. 5. The user clicks on the “Predict” button to submit the symptoms for prediction. 6. The system validates that at least one symptom is selected.
Sub-flows: -
Alternate/Exceptional Flows: 6a. If the user clicks “Predict” without selecting or entering at least one symptom in the checklist or free text input box, the system displays “Please select at least one symptom to proceed.” message. 6b. If the input is valid, the system executes the “View Prediction Results” use case.

4.3.2.4 View Predicted Results

Table 4.6: Use case description of View Predicted Results

Use Case Name: **View Predicted Results**		ID: **UC004**	Importance Level: **High**
Primary Actor: **User**	Use Case Type: **Details, Essential**		
Stakeholders and Interests: **User – wants to view the predicted results based on the input symptoms.**			
Brief Description: **This use case describes how a user views the predicted results after inputting symptoms into the system.**			
Trigger: **The user wants to view the predicted results after completing the “Input Symptoms” use case.**			

Relationships: Association : **User** Include : **-** Extend : **View Medical Advice** Generalization**: -**
Normal Flow of Events**:** 1. The system processes the input symptoms using the ML model. 2. The system displays prediction results. 3. The user views the prediction results on screen. 4. The system displays 2 buttons for back and viewing medical advice.
Sub-flows**: -**
Alternate/Exceptional Flows: 2a. The system displays "Unable to process prediction. Please try again later." message if the prediction fails due to server or model issues. 3a. The system displays "No specific condition matched. Please refine your input." message if the result is not found.

4.3.2.5 View Medical Advice

Table 4.7: Use case description of View Medical Results

Use Case Name**: View Medical Advice**	ID**:** **UC005**	Importance Level**:** **High**
Primary Actor**: User**	Use Case Type**: Details, Essential**	
Stakeholders and Interests**:** **User – wants to view the medical advice related to the predicted diseases for further action.**		
Brief Description**: This use case describes how a user views medical advice after receiving the disease prediction results.**		

Trigger**: The user wants to seek medical advice after receiving the disease prediction results.**
Relationships: Association **: User** Include **: -** Extend **: -** Generalization**: -**
Normal Flow of Events**:** 1. The system displays the "View Advice" button. 2. The user clicks on "View Advice" button. 3. The system displays medical advice for the user related to predicted diseases. 4. The system displays 2 buttons for back and storing the prediction results and medical advice.
Sub-flows**: -**
Alternate/Exceptional Flows: 3a. The system display "No advice available for this condition at this time." message if there is no advice is available for the prediction.

4.3.2.6 Store Predicted Results

Table 4.8: Use case description of Store Predicted Results

Use Case Name**: Store Predicted Results**		ID**:** **UC006**	Importance Level**:** **High**
Primary Actor**: User**	Use Case Type**: Details, Essential**		
Stakeholders and Interests**: User – wants to store the prediction result for future reference**			

Brief Description**: This use case describes how a user stores the prediction results into the system.**
Trigger**: The user wants to store the input symptoms, prediction results and medical advice into database.**
Relationships: Association **: User** Include **: -** Extend **: -** Generalization**: -**
Normal Flow of Events**:** 1. The user clicks on the "Store Results" button in the view medical advice page. 2. The system stores the input symptoms, prediction results and medical advice to the database. 3. The system displays successful messages when the data is successful store in the database.
Sub-flows**: -**
Alternate/Exceptional Flows: 3a. The system displays the error message "Fails to save result. Please try again" if the storage data failure occurs.

4.3.2.7 View Historical Results

Table 4.9: Use case description of View Historical Results

Use Case Name**: View Historical Results**	ID: **UC007**	Importance Level**: High**

Primary Actor: **User**	Use Case Type: **Details, Essential**
Stakeholders and Interests: **User – wants to view the past prediction results to track the symptoms over time.**	
Brief Description: **This use case describes how a user views their past disease prediction results to monitor the trends.**	
Trigger: **The user wants to view the historical information by clicking the "View History" button.**	
Relationships: Association **: User** Include **: -** Extend **: -** Generalization**: -**	
Normal Flow of Events: 1. The user selects the "My History" option in the home page. 2. The system retrieves the past prediction results of users from the database. 3. The system displays a list of the past prediction results including symptoms input. 4. The user selects the specific result to view. 5. The system displays detailed prediction results of the selected result.	
Sub-flows:	
Alternate/Exceptional Flows: 1a. If the user is not yet logged in, the system prompts the user to log in. 2a. The system displays "No prediction history found" message if the record is not found.	

4.3.2.8 View Profile

Table 4.10: Use case description of View Profile

<table>
<tr><td colspan="2">Use Case Name: View Profile</td><td>ID: UC008</td><td>Importance Level: High</td></tr>
<tr><td>Primary Actor: User</td><td colspan="3">Use Case Type: Details, Essential</td></tr>
<tr><td colspan="4">Stakeholders and Interests:
User – wants to view their personal data.</td></tr>
<tr><td colspan="4">Brief Description: This use case describes how a user views their personal data.</td></tr>
<tr><td colspan="4">Trigger: The user wants to view the profile.</td></tr>
<tr><td colspan="4">Relationships:
Association : User
Include : -
Extend : Update Profile
Generalization: -</td></tr>
<tr><td colspan="4">Normal Flow of Events:
1. The system displays the “My Profile” button in the header in Home Page.
2. The user clicks on “My Profile” button.
3. The system displays the user's personal information, such as e-mail address, date of birth and gender, on the profile page. Gender is empty by default unless the user wants to update it.
4. The system provides a link for the user to change their password.
5. The user enters the current password, new password and confirmation password.
6. The system checks and validates of password. Perform 6.1 or 6.2.
6.1 If the passwords are valid, flow no.7 continues.
6.2 If the passwords are invalid, the system will indicate that the changes were unsuccessful, the flow no.4 continues.
7. The user successfully changes their password.</td></tr>
</table>

Sub-flows: -
Alternate/Exceptional Flows:

4.3.2.9 Update Profile

Table 4.11: Use case description of Update Profile

Use Case Name: **Update Profile**		ID: **UC009**	Importance Level: **High**
Primary Actor: **User**	Use Case Type: **Details, Essential**		
Stakeholders and Interests: **User – wants to update their personal data.**			
Brief Description: **This use case describes how a user updates their personal data.**			
Trigger: **The user wants to update the profile.**			
Relationships: Association **: User** Include **: -** Extend **: -** Generalization**: -**			
Normal Flow of Events: 1. The system displays the "Update Profile" button in the header in Profile Page. 2. The user clicks on "Update Profile" button to edit their personal data. 3. The system displays update forms for user including username, date of birth and gender. 4. The user enters the new personal information. 5. The system checks and validates the information. *Perform 5.1 or 5.2.*			

5.1 If the data are valid, **flow no.6** continues. 5.2 If the data are invalid, the system will indicate that the update was unsuccessful, the flow no.1 continues. 6. The user successfully updates their profile.
Sub-flows**: -**
Alternate/Exceptional Flows:

4.4 System Flow Diagram

The Figure 4.2 shows the system flow diagram for the disease prediction web application using machine learning system.

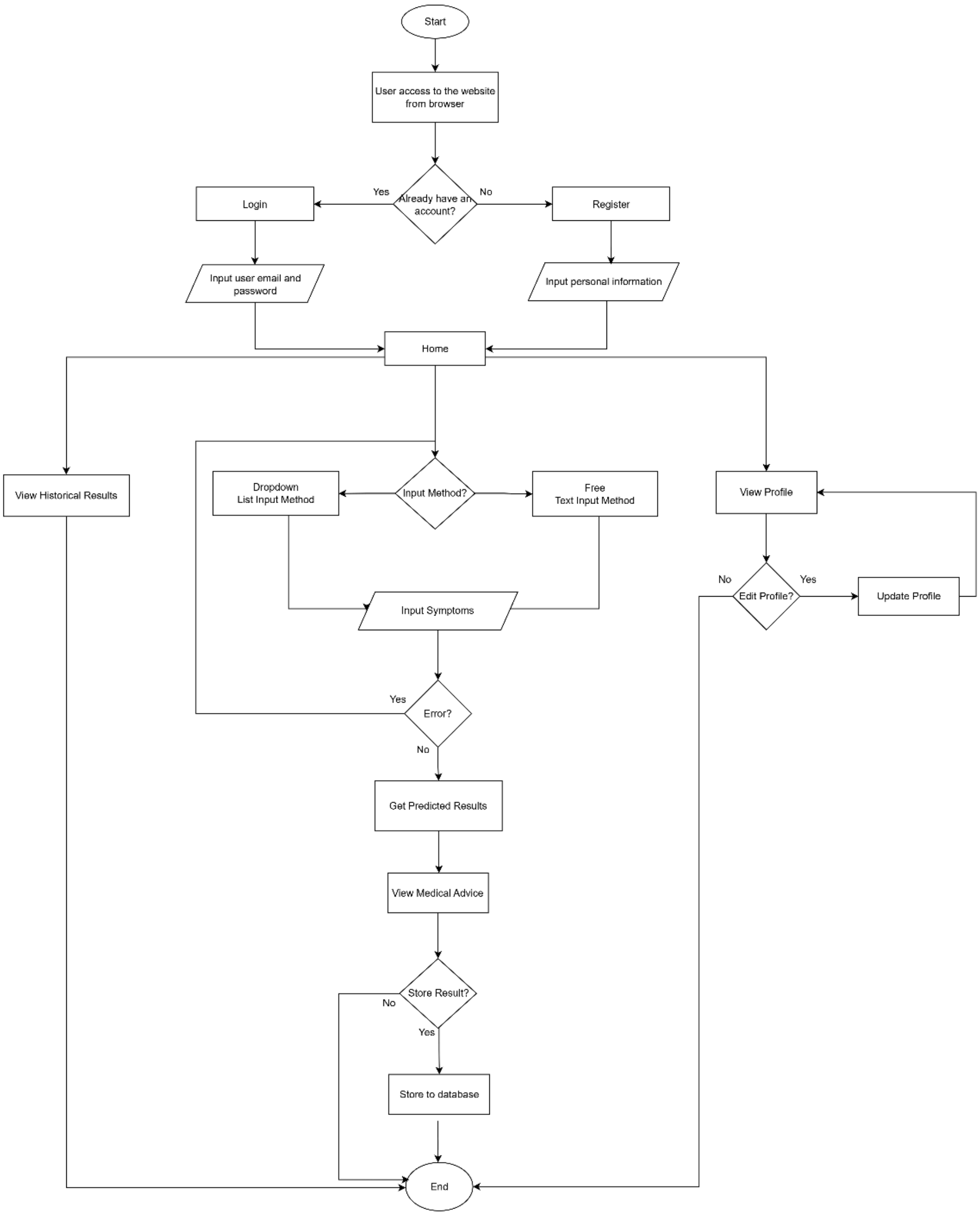


Figure 4.2: System Flow Diagram

4.5 Interface Flow Diagram

The Figure 4.3 shows the interface flow diagram of Disease Prediction Web Application using Machine Learning.

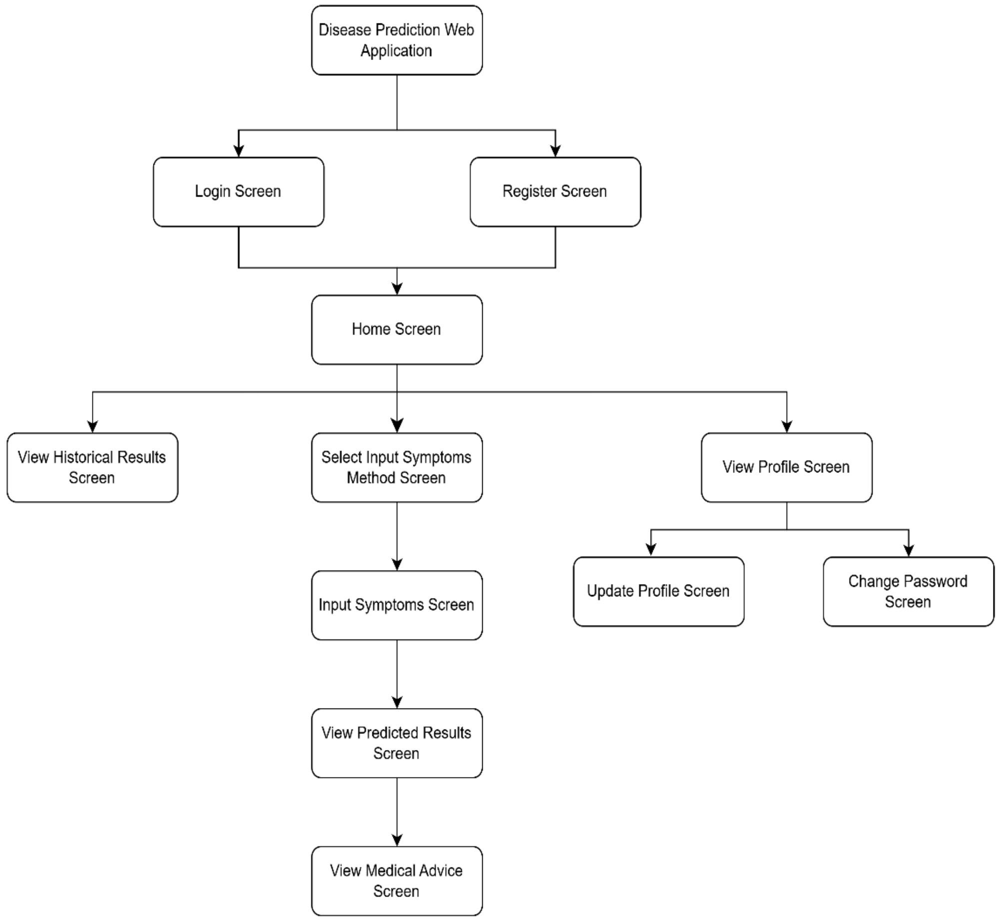


Figure 4.3: Interface Flow Diagram

4.6 Low Fidelity Prototype

4.6.1 Welcome Page

The welcome page is the first page of Disease Prediction Web Application using Machine Learning. Welcome page includes login and register buttons for user to access the web application.

Figure 4.4: Welcome Page

4.6.2 Login Page

The login page allows user to input their email address and password.

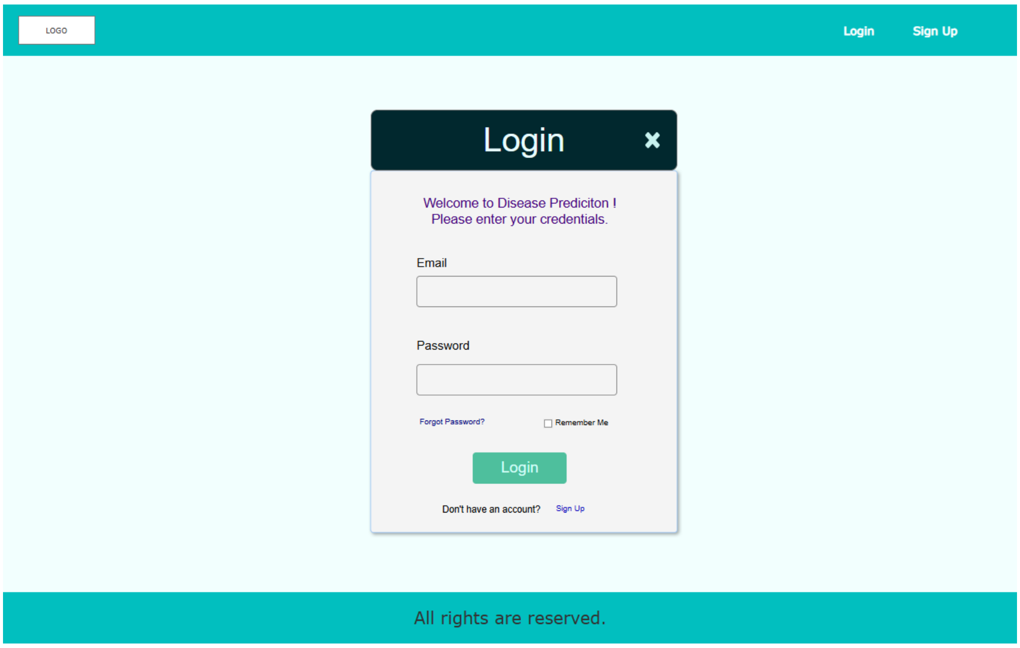


Figure 4.5: Login Page

4.6.3 Sign Up Page

The sign up (register) page allows users to create a new account.

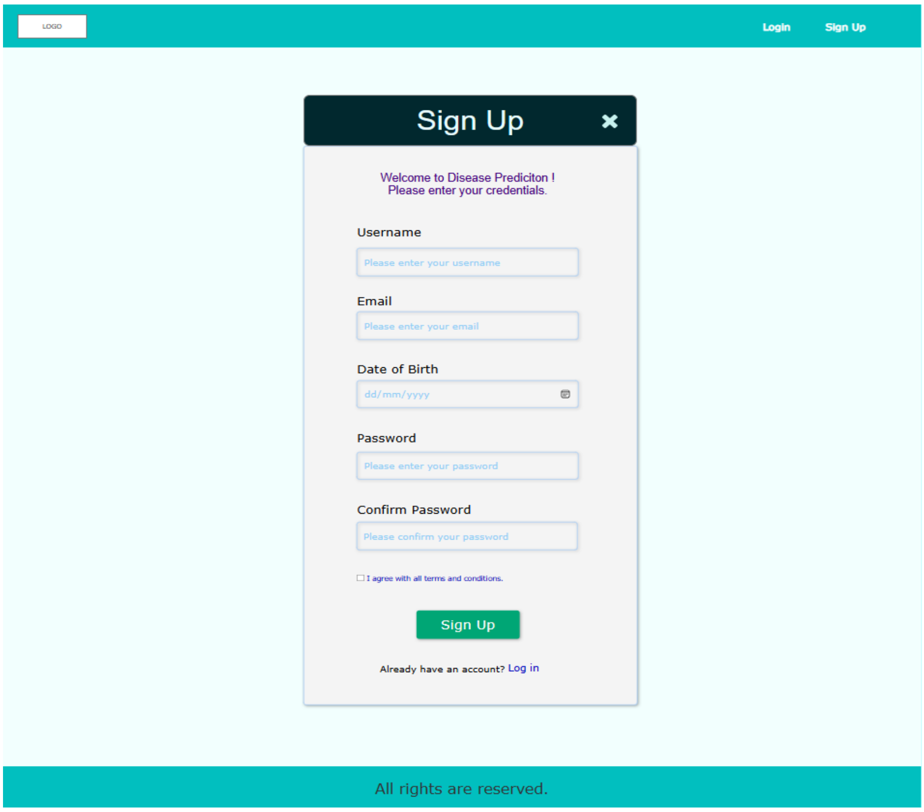


Figure 4.6: Sign Up Page

4.6.4 Home Page

The Home Page allows users to access to view the history, profile and displays the username in the header. The Home Page also allows users to choose the input methods to predict the potential diseases. The Home Page briefly describes about how the web application works.

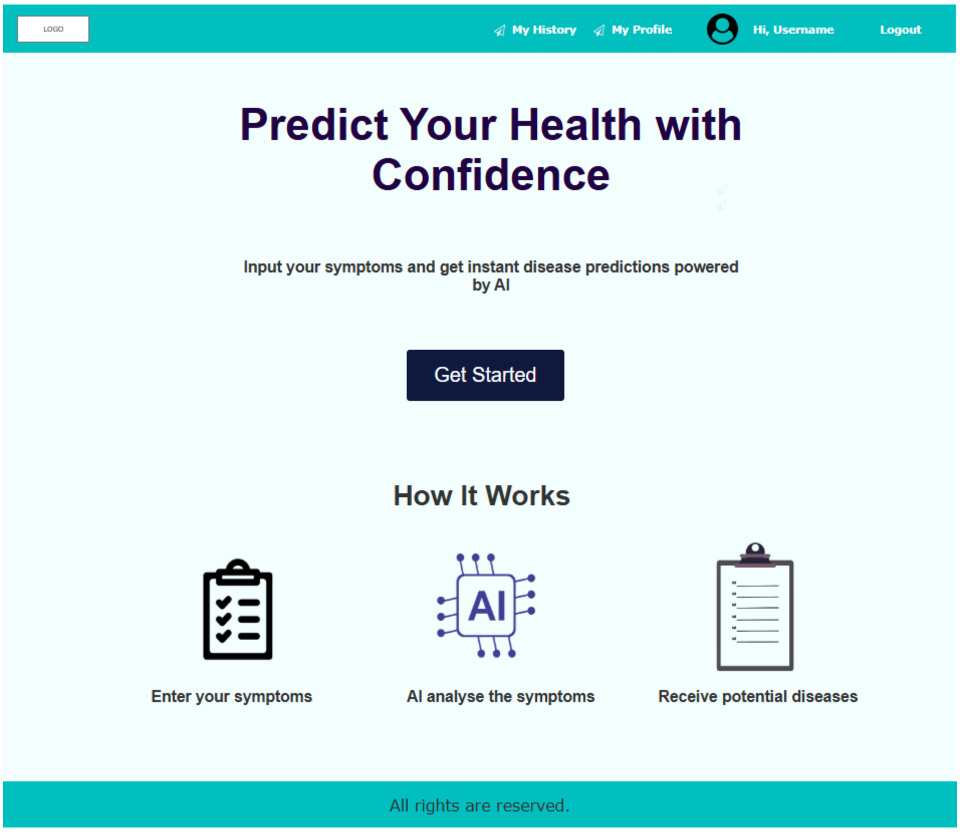


Figure 4.7: Home Page

4.6.5 Select Input Method Page

The Select Input Method Page allows users to choose the input methods to predict the potential diseases.

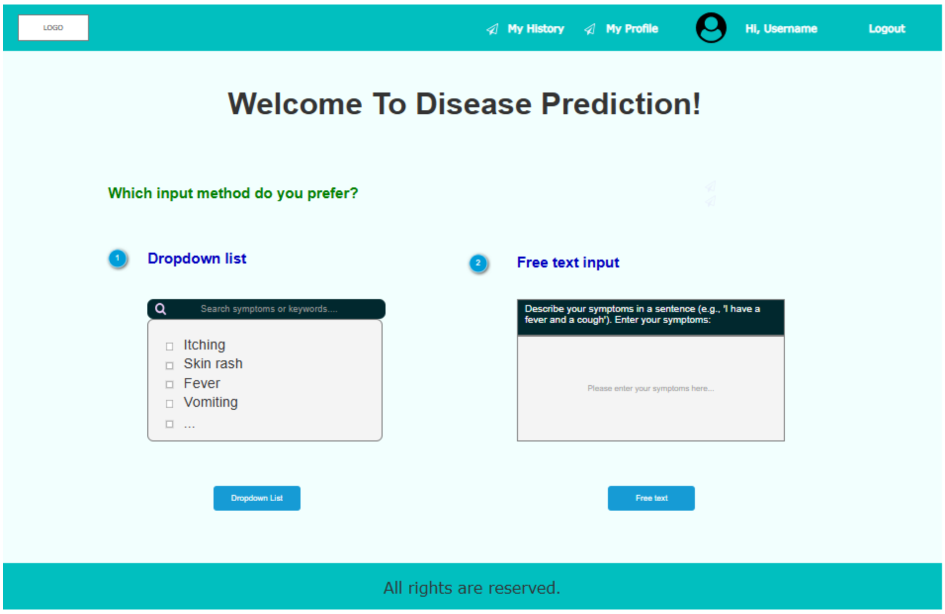


Figure 4.8: Select Input Method Page

4.6.6 Dropdown List Input Symptoms Page

The Dropdown List Input Symptoms Page allows user to search and select the relevant symptoms they may have, and the "Add" button add the symptoms to the "Selected Symptoms". For the "Selected Symptoms", user also allowed to delete the symptoms. If a complete symptom is entered and click the "Prediction" button, the system starts the prediction.

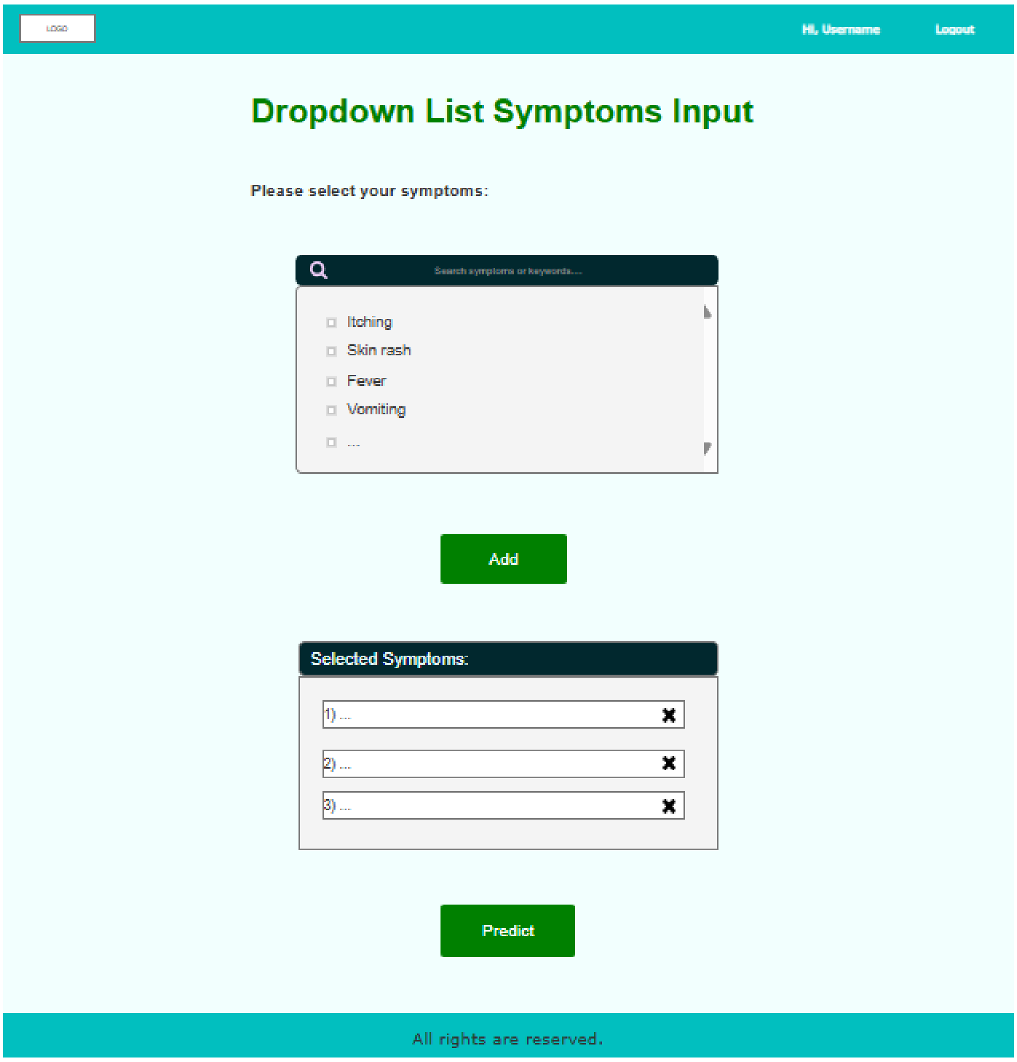


Figure 4.9: Dropdown List Input Symptoms Page

4.6.7 Free Text Input Symptoms Page

The Free Text Input Symptoms Page allows user to enter the relevant symptoms they may have, and the "Add" button add the symptoms, the system will display the "Matched Symptoms". For the "Matched Symptoms", user also allowed to delete the symptoms. If a complete symptom is entered and click the "Prediction" button, the system starts the prediction.

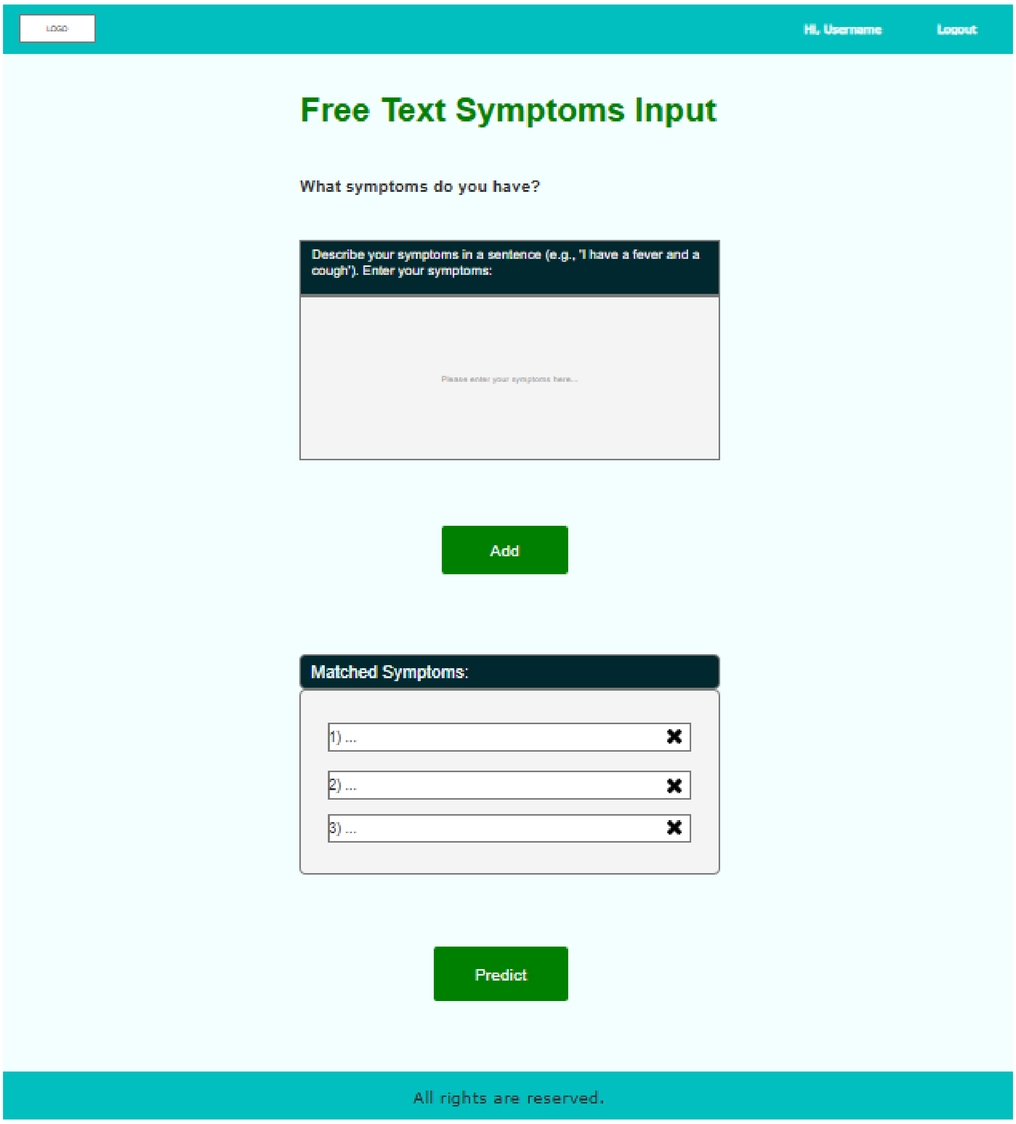


Figure 4.10: Free Text Input Symptoms Page

4.6.8 Predicted Results Page

The Predicted Results page shows potential diseases that match the user's symptoms. The right side shows the symptoms that the user entered into the system.

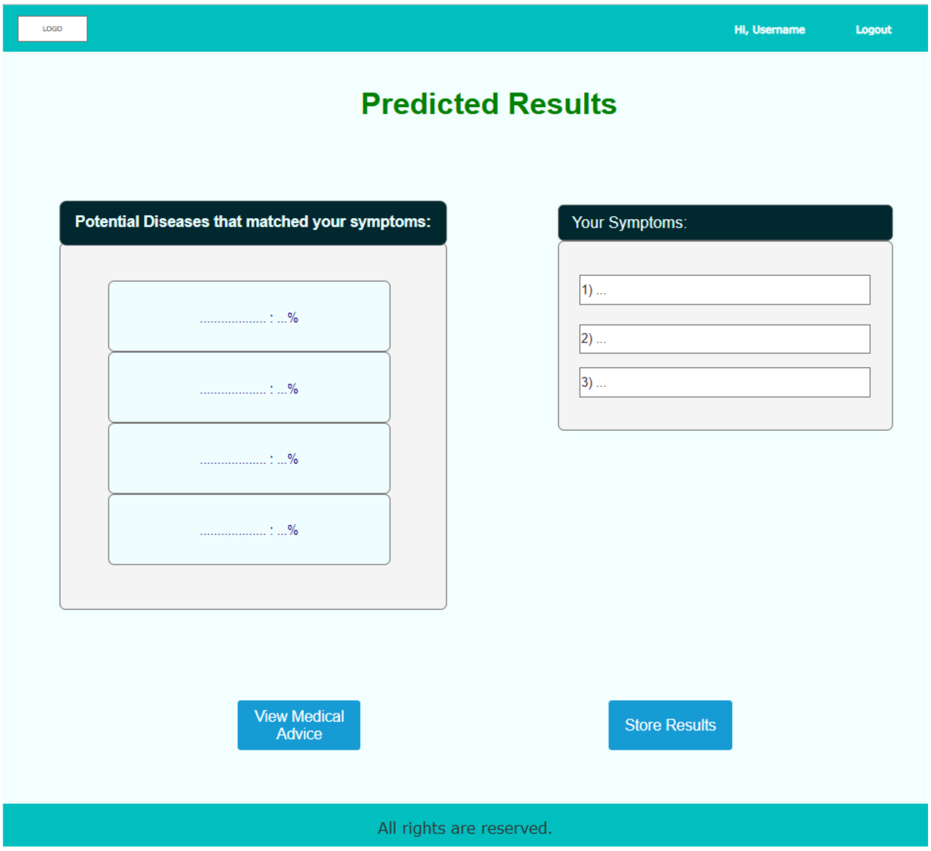


Figure 4.11: Predicted Results Page

4.6.9 View Medical Advice Page

This page is for user to view the medical advice based on predicted results.

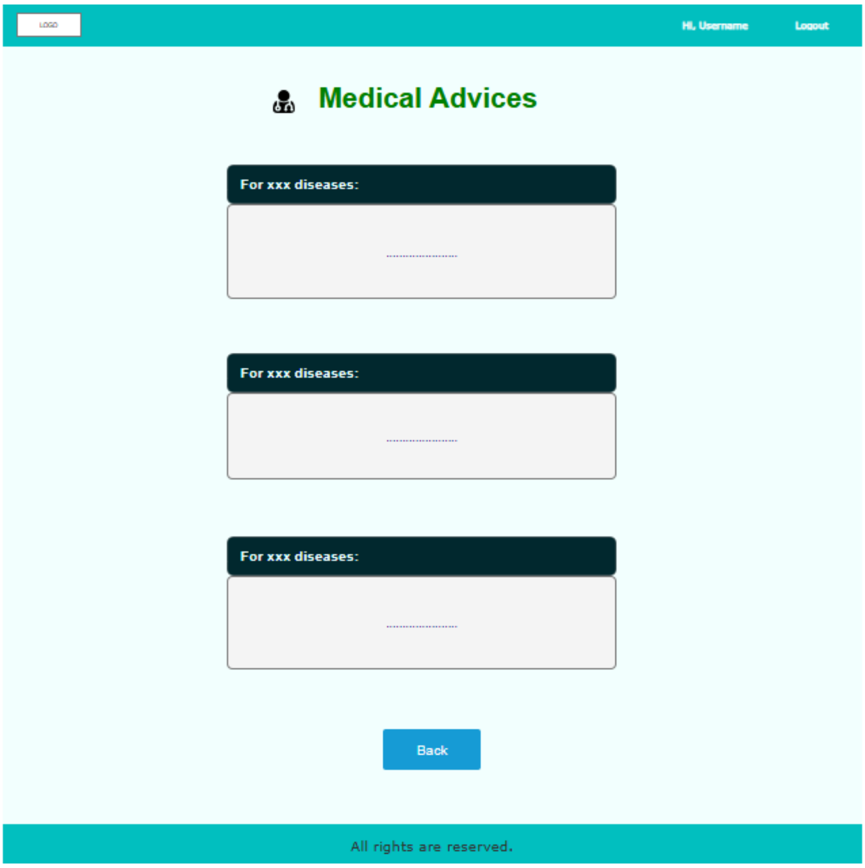


Figure 4.12: View Medical Advice Page

4.6.10 View Historical Results Page

This page is for user to view their historical results in the system.

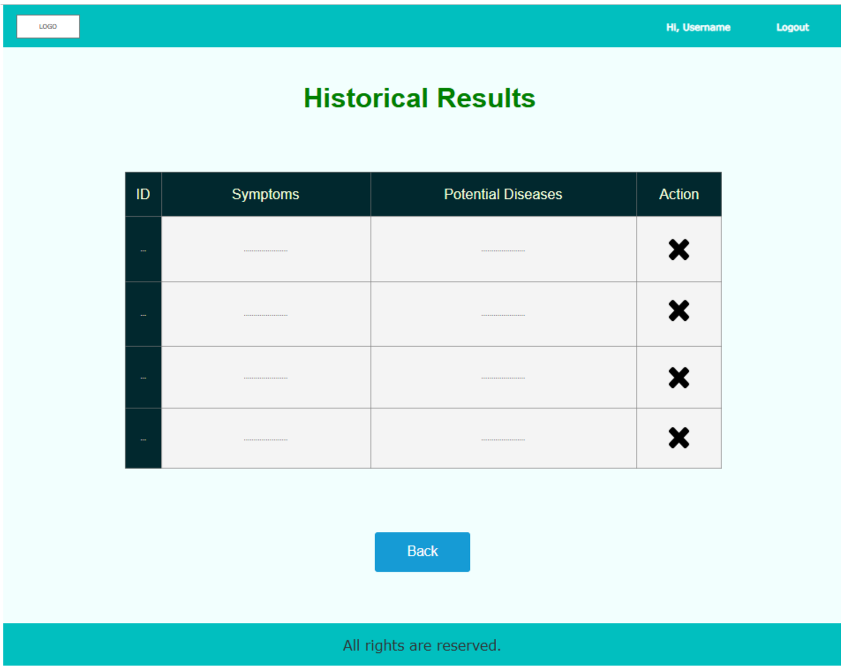


Figure 4.13: View Historical Results Page

4.6.11 Profile Page

The profile page allows users to view their personal information on the profile page. Gender is empty by default unless the user wants to update it. Users can change their password by following the link.

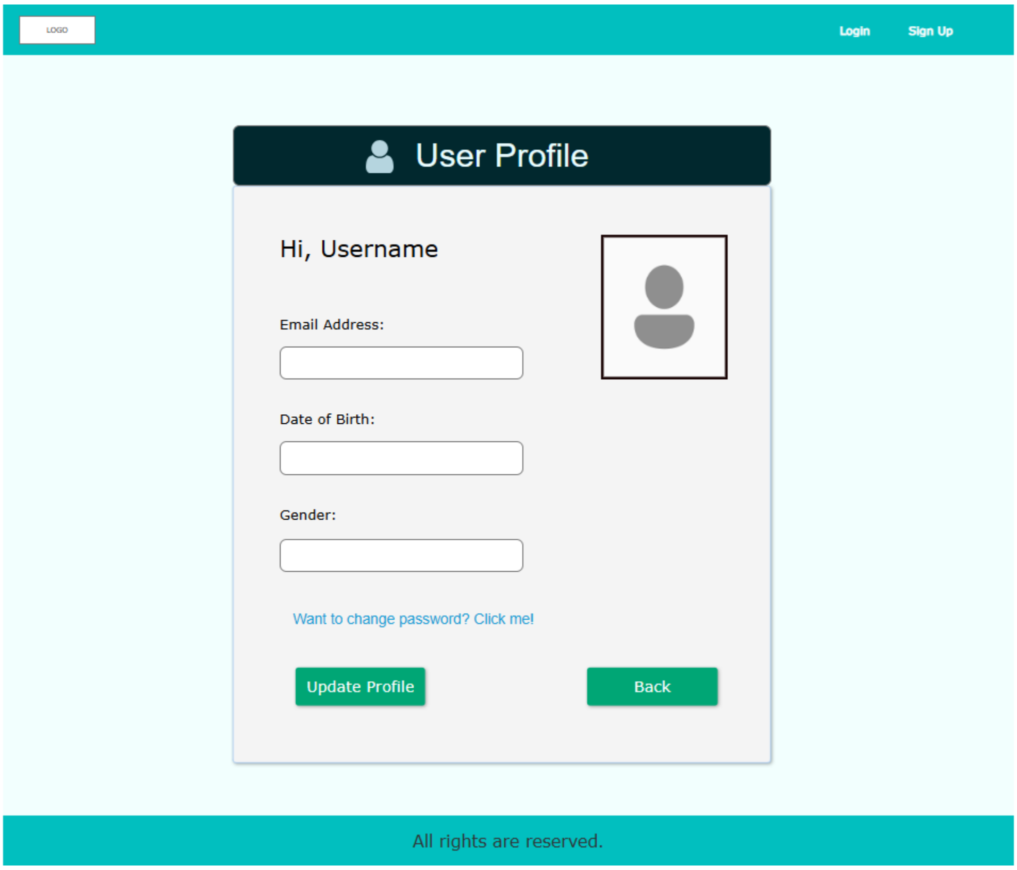


Figure 4.14: Profile Page

4.6.12 Update Profile Page

This page allows users to update their personal information such as username, date of birth, and gender.

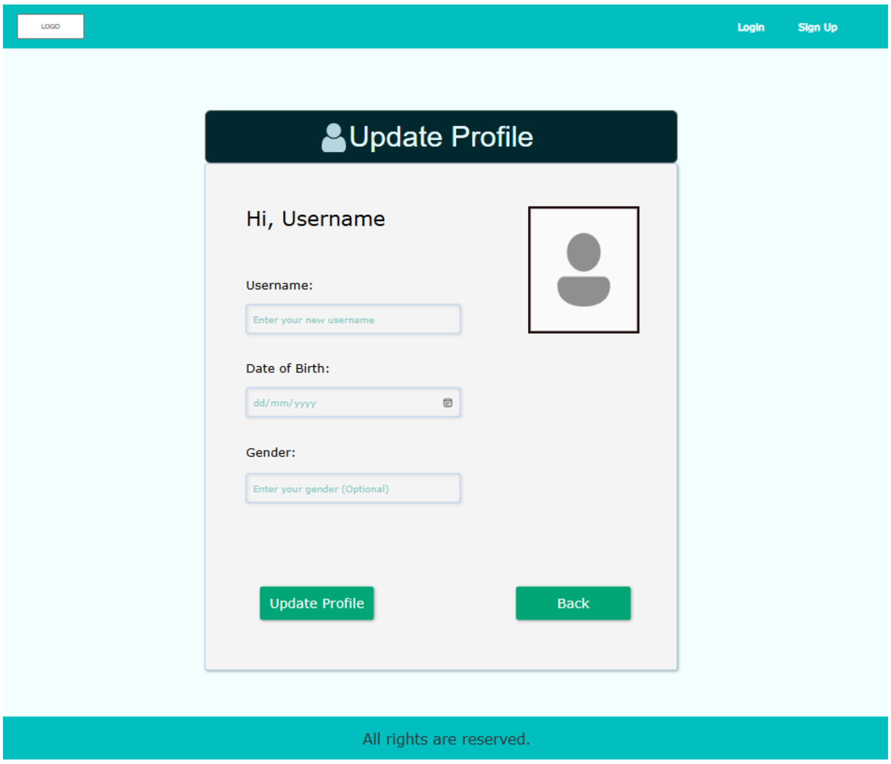


Figure 4.15: Update Profile Page

4.6.13 Change Password Page

This page allows user to change their password.

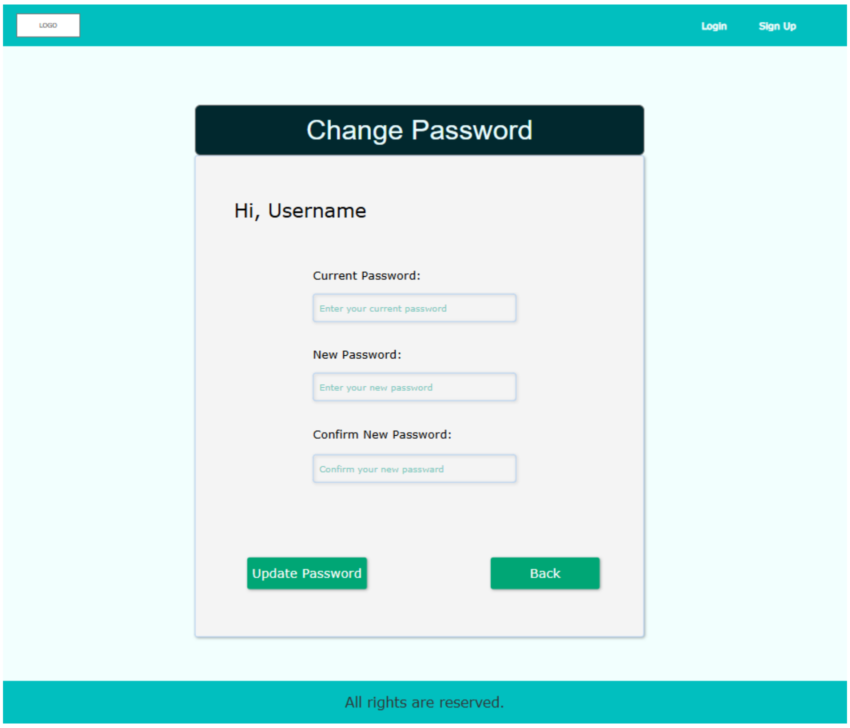


Figure 4.16: Change Password Page

4.7 Preliminary Results

This section is to present the preliminary results obtained during the early implementation and testing phases of the Random Forest model building. These results validate the feasibility of the proposed system design. This section focuses on the evaluate the capabilities of the Random Forest model, functional requirements of dropdown list input system and free-text input system.

4.7.1 Model Performance

The Random Forest model was developed and trained on the Symptom-Disease Prediction Dataset (SDPD) dataset, which includes 132 symptoms and cover 41 diseases with a total of 4920 instances. The dataset is split into two subsets which are training, validation testing sets in a ratio of 80-20, resulting in 3936 training samples and 984 testing samples. This model was trained with 100 trees (n_estimators=100) and 42 random states for reproducibility. The n_estimators=100 is the number of decision trees built, and each tree is trained on a random subset of the features and data. Prediction is combined with majority voting to improve the accuracy and reduce the overfitting. The random_state =42 parameter is used to assign a seed to the random number generator, ensuring that the train-test split, as well as the tree construction are the same each time the model is trained, thus making the results are reproducible. The model was evaluated using Accuracy, sensitivity (recall), and F1-Score. The overall metrics in this example are not accurate and are for demonstration purposes only. The Figure 4.12 shows the preliminary Random Forest model performance metrics. The Figure 4.13 shows the sample of Pre-Disease performance metrics.

```
Overall Metrics:

Model Accuracy on Test Set: 100.00%
Macro Average Recall (Sensitivity): 1.00
Macro Average F1-Score: 1.00
```

Figure 4.17: Preliminary Model Performance Metrics

```
Detailed Metrics by Disease:
Disease                     Precision   Recall (Sensitivity)F1-Score    Support
--------------------------------------------------------------------------------
AIDS                        1.00        1.00                1.00        29
Acne                        1.00        1.00                1.00        25
Alcoholic Hepatitis         1.00        1.00                1.00        29
```

Figure 4.18: Sample of Per-Disease Performance Metrics

4.7.2 Dropdown List Input

One of the functional requirements of the project is the system allow users to input their symptoms by selecting the symptoms from the predefined list. This method presents the user with the complete list of 132 symptoms in the dataset and allows the user to select symptoms by typing the exact names. Figure 4.14 shows the sample of the dropdown list input and the prediction outputs. The output may not be accurate and was improved in the next chapter.

```
Choose your input method:
1. Select symptoms from the list (type each symptom)
2. Free-text input (type a sentence describing your symptoms)
Enter 1 or 2: 1

Enter your symptoms (type the symptom name exactly as shown in the list above).
When done, type 'done' to get the prediction.

Enter a symptom (or 'done' to finish): itching
Added symptom: itching
Enter a symptom (or 'done' to finish): skin_rash
Added symptom: skin_rash
Enter a symptom (or 'done' to finish): done

Top 5 Potential Diseases (with probabilities):
Fungal Infection: 53.00%
Drug Reaction: 21.00%
Acne: 6.00%
Chronic Cholestasis: 4.00%
Cervical Spondylosis: 3.00%
```

Figure 4.19: Sample Dropdown List Input and Prediction Results

4.7.3 Free-Text Input

One of the functional requirements of the project is the system allow users to input their symptoms by using free text input method. By using Google Gemini to extract symptoms, this approach effectively handles the spelling errors and text issues. The free text input method maps the user input to the 132 symptoms in the dataset through the synonym dictionary. Figure 4.15 shows the sample of free-text input and prediction output. The output may not be accurate and was improved in next chapter.

```
Choose your input method:
1. Select symptoms from the list (type each symptom)
2. Free-text input (type a sentence describing your symptoms)
Enter 1 or 2: 2

Describe your symptoms in a sentence (e.g., 'I have a fever and a cough').
Enter your symptoms: I have a fever, I'm coughing, my nose is running, and I feel tired
Matched symptom: cough
Matched symptom: fatigue
Matched symptom: runny_nose
Matched symptom: high_fever

Top 5 Potential Diseases (with probabilities):
Bronchial Asthma: 49.00%
Common Cold: 8.00%
Impetigo: 8.00%
Varicose Veins: 8.00%
AIDS: 6.00%
```

Figure 4.20: Sample of Free-Text Input and Prediction Output

CHAPTER 5

SYSTEM DESIGN

5.1 Introduction

This chapter demonstrates the system design for the Disease Prediction Web Application. The system design encompasses the system architecture, data model design, database design, entity relationship diagram (ERD), user interface design and prompt design. This can ensure that the final implementation is both efficient and user-friendly. Moreover, the last section presents a high-fidelity prototype of the user interface as a reference for the interface design of the system.

5.2 System Architecture Design

The system architecture defines the overall structure of the web application. The proposed disease prediction web application adopts a three-tier architecture, comprising the frontend, backend, and database, with the machine learning model and large language model (Google Gemini) integrated into the backend. The Figure 5.1 shows the three-tier architecture of this project. The Figure 5.2 shows the System Architecture Design to clarify the behavioural patterns and operational structure of the system.

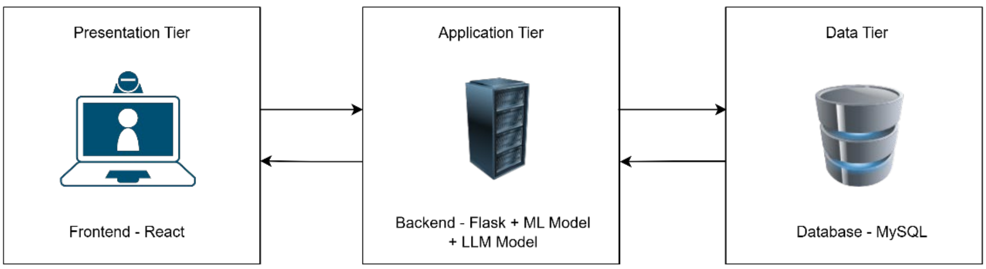


Figure 5.1: Three-tier Architecture Diagram

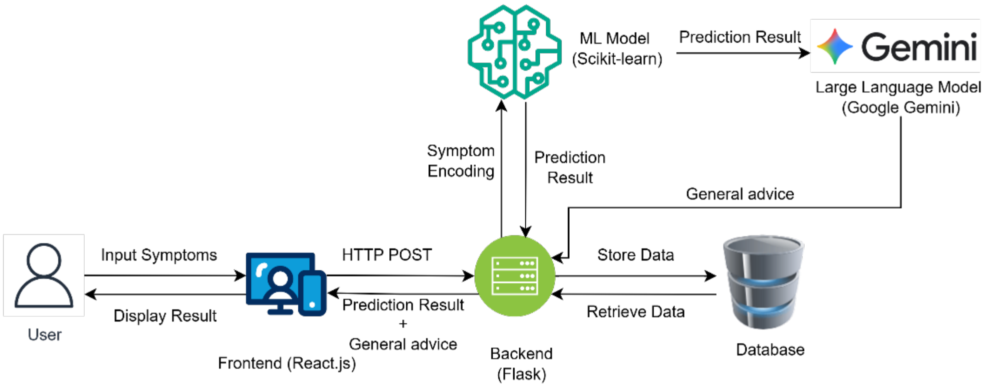


Figure 5.2: System Architecture Design

The system employs a client-server architecture based on modular design to ensure maintainability and scalability while enabling the system to handle diverse request type efficiently. This architecture is divided into three main tiers, including Presentation Tier, Application Tier, and Data Tier. This layered approach achieves separation of concern, enabling each component to be developed, tested, and extended independently. This architecture is designed to efficiently process user request, analyse symptoms through machine learning model, deliver predictive results in real time, and utilize large language models to extract possible symptoms from users' free-text inputs while provide general medical advice.

The presentation tier serves as the user interface and user experience (UI/UX). It enables users to register, login, input symptoms, view prediction results, check general medical advice, and review their history. The system using React to build dynamic and responsive frontend, utilises external CSS for styling and supports RESTful API calls. React was chosen for its efficient handling of dynamic updates, delivering a smooth user experience.

For the application tier, the backend is implemented using Flask. As s lightweight Python framework, Flask serves as the middle tier between the frontend and the data tier. It handles user request, manages user session authentication, validates user input, and communicates with the database. This tier communication with the frontend by exposing RESTful API endpoints such as /predict, /user. Both machine learning model and large language model are integrated into this tier. The machine learning model is developed using scikit-learn and is responsible for processing the input symptoms and generate predictions by comparing them against patterns learned from training data. The large language model is using Google Gemini API to extract free-text symptoms inputs and generate general medical advice based on different diseases and return it as a JSON response. The backend ensures timely return of predictions results and supplemented by relevant medical advice.

The data tier is used to manage the persistent storage of user data. This tier is implemented by using MySQL, which stores user credentials and information, symptom and disease records, prediction history, and general medical advice. The database ensures consistency, integrity and security of stored information as well as efficiently managing queries and updates to support real-time interaction with the application.

This architecture is scalable as improvements on one layer can be implemented independently without affecting the others. Furthermore, the modularity of the system enables the system to seamless integration of additional features such as advanced medical knowledge bases for generating medical recommendations.

5.3 Data Model Design

This section outlines the database design for the Disease Prediction Web Application using Machine Learning. The database is designed to efficiently store and manage the application data, ensuring quick retrieval and secure storage. The design includes the conceptual data model including entities and relationships and its physical implementation in MySQL. MySQL was selected as the relational database management system due to its robustness, high performance, and widespread use.

5.3.1 Entity Relationship Diagram (ERD)

The ERD visually represents the relationship between entities, as shown in Figure 5.3.

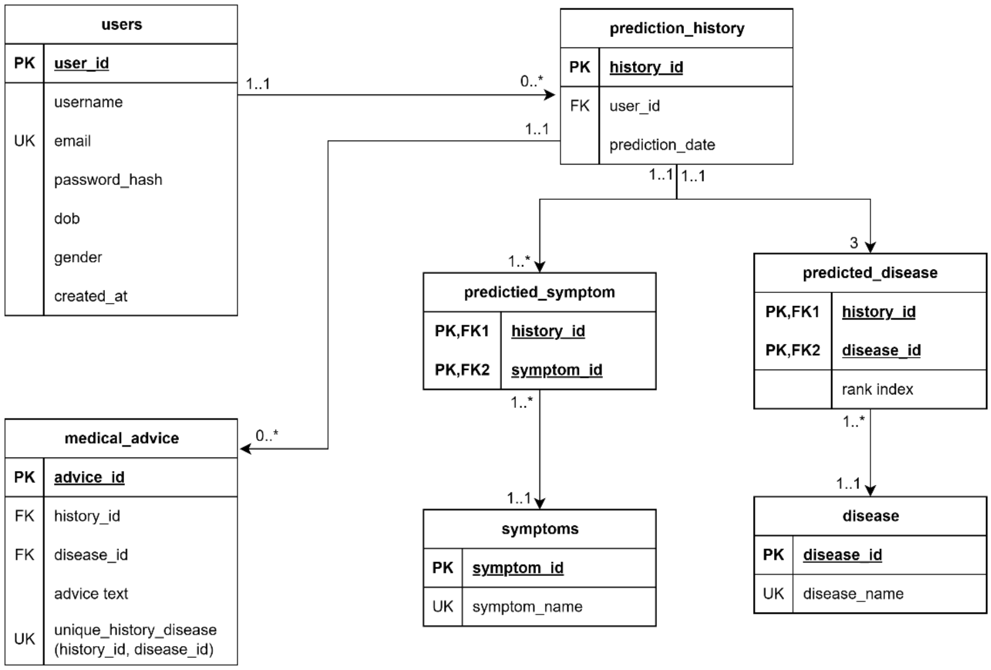


Figure 5.3: Entity Relationship Diagram (ERD)

5.3.2 Data Dictionary

The following data dictionary provides a detailed description of each table, including attributes, data types, and constraints.

Table 1: users

Column Name	Data Type	Description	Constraints
user_id	INT	Unique identifier for each user	Primary Key, Auto increment
username	VARCHAR (191)	Login name of user	Not Null
email	VARCHAR (191)	Email address of user	Unique, Not Null
password_hash	VARCHAR (255)	Hashed password for secure login	Not Null
dob	DATE	Date of birth of user	Optional
gender	VARCHAR (10)	Gender of user	Optional

created_at	TIMESTAMP	Account creation timestamp	Default: CURRENT_ TIMESTAMP

Table 5.1: users Table Data Dictionary

Table 2: symptoms

Column Name	Data Type	Description	Constraints
symptom_id	INT	Unique identifier for symptom	Primary Key, Auto Increment
symptom_name	VARCHAR (191)	Symptom name	Unique, Not Null

Table 5.2: symptoms Table Data Dictionary

Table 3: diseases

Column Name	Data Type	Description	Constraints
disease_id	INT	Unique identifier for disease	Primary Key, Auto Increment
disease_name	VARCHAR (191)	Disease name	Unique, Not Null

Table 5.3: diseases Table Data Dictionary

Table 4: prediction_history

Column Name	Data Type	Description	Constraints
history_id	INT	Unique identifier for prediction session	Primary Key, Auto Increment
user_id	INT	User who made the prediction	Foreign Key (users – user_id)
prediction_date	TIMESTAMP	Date and time of prediction	Default: CURRENT_ TIMESTAMP

Table 5.4: prediction_history Table Data Dictionary

Table 5: medical_advice

Column Name	Data Type	Description	Constraints
advice_id	INT	Unique identifier for advice record	Primary Key, Auto Increment
history_id	INT	Related prediction session	Foreign Key (prediction_history – history_id), ON DELETE CASCASE
disease_id	INT	Related disease	Foreign Key (disease – disease_id)
advice_text	TEXT	General medical advice gets from LLM	Not Null
unique_history_disease	constraint	Ensures only one advice per (history, disease) pair	Unique

Table 5.5: medical_advice Table Data Dictionary

Table 6: predicted_symptom (junction table)

Column Name	Data Type	Description	Constraints
history_id	INT	Prediction session ID	Foreign Key (prediction_history – history_id)
symptom_id	INT	Symptom included in this prediction session	Foreign Key (symptoms – symptom_id)
Primary Key	(history_id, symptom_id)	Composite primary key	

Table 5.6: predicted_symptom junction table Data Dictionary

Table 7: predicted_disease (junction table)

Column Name	Data Type	Description	Constraints
history_id	INT	Prediction session ID	Foreign Key (prediction_history – history_id)
disease_id	INT	Disease predicted	Foreign Key (diseases – disease_id)
rank_index	INT	Rank of prediction	Default 0
Primary Key	(history_id, disease_id)	Composite primary key	

Table 5.7: predicted_disease junction table Data Dictionary

5.4 User Interface Design

The user interface (UI) of the disease prediction web application using machine learning prioritizes on simplicity, ease of use, and accessibility during development, ensuring that users with varying levels of technical expertise can effectively interact with the system. The UI design is focussed on creating an intuitive interface that enables users to input symptoms, view disease prediction results, and achieve seamless interaction with the system. The frontend is implemented using React, leveraging component-based design to promote modularity and reusability while maintaining consistency across different pages of the application.

5.4.1 Welcome Page

Upon launching the web application, users are greeted with a welcome interface, featuring login and registration button to facilitate secure access to the system. The header simultaneously displays the system logo, login portal and register portal to users.

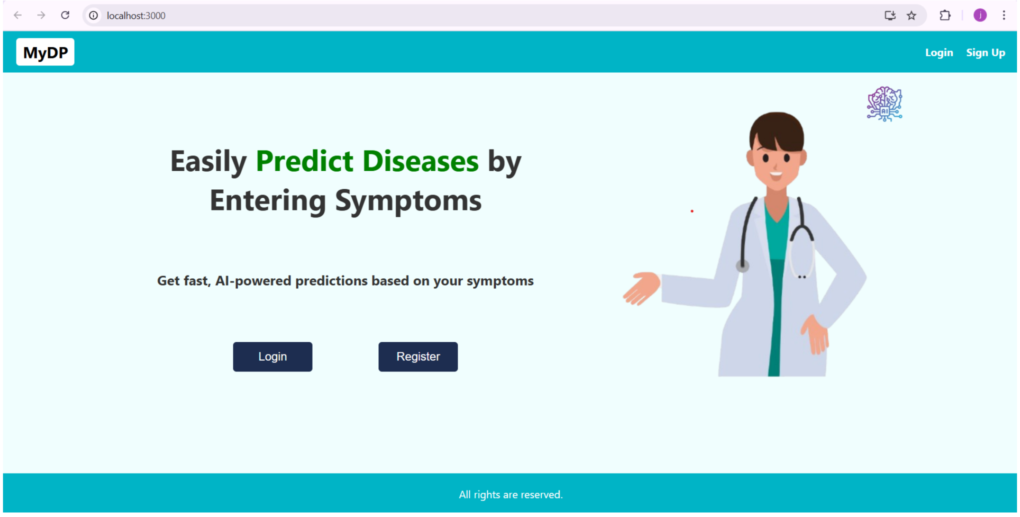


Figure 5.4: Actual Welcome Page

5.4.2 Login Page

The Login page allows users to access to the system. Once authenticated, the user is directed to the Home Page, providing access to the core features such as symptom input and so on.

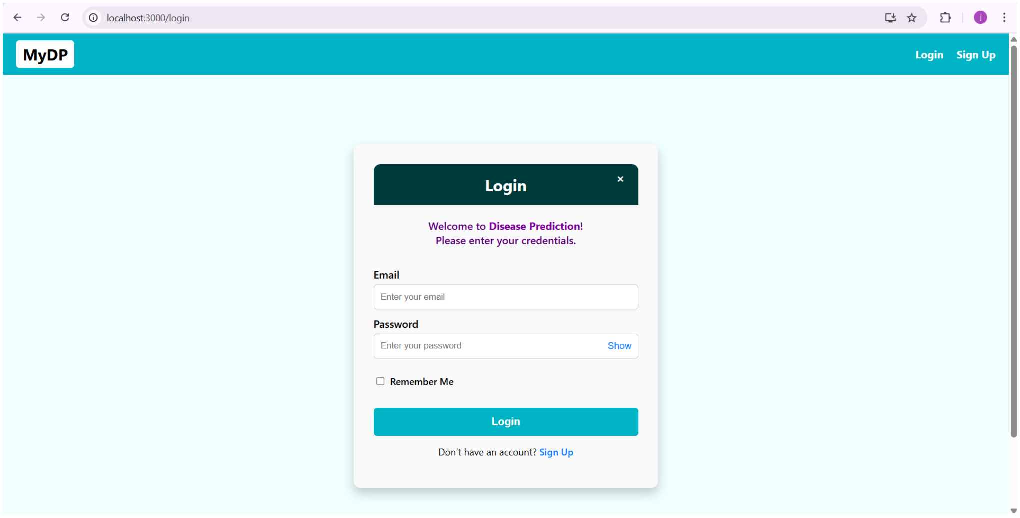


Figure 5.5: Actual Login Page

5.4.3 Sign Up Page

The Sign Up Page enables users to create account and access the system securely. This page is designed with minimalistic form structures with clear field labels, and real-time validation feedback to reduce user input errors.

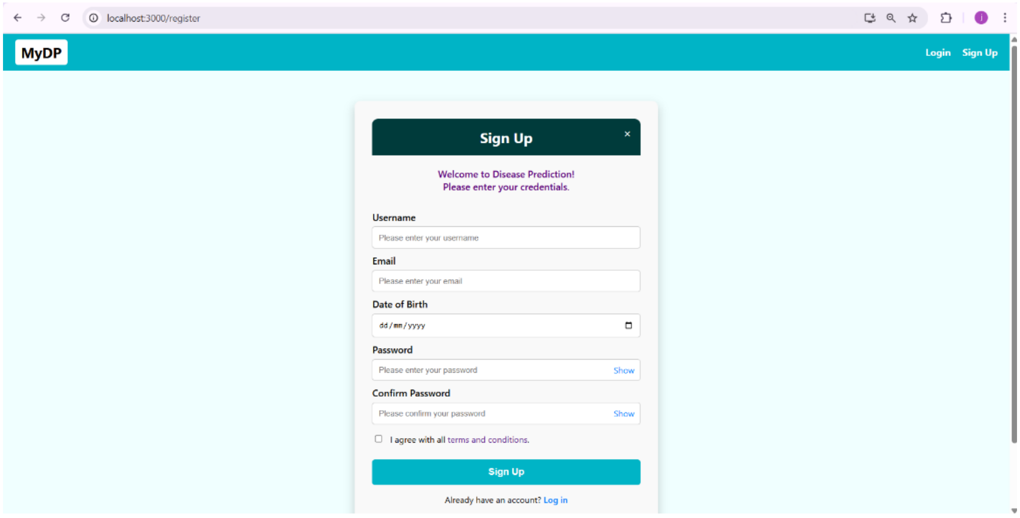


Figure 5.6: Actual Sign Up Page

5.4.4 Home Page

The Home Page provides an overview of the application. It includes a brief description of the disease prediction system. A "Get Started" button to navigate to the prediction input form. Navigation bar with Home, History, Profile, Username and Logout button.

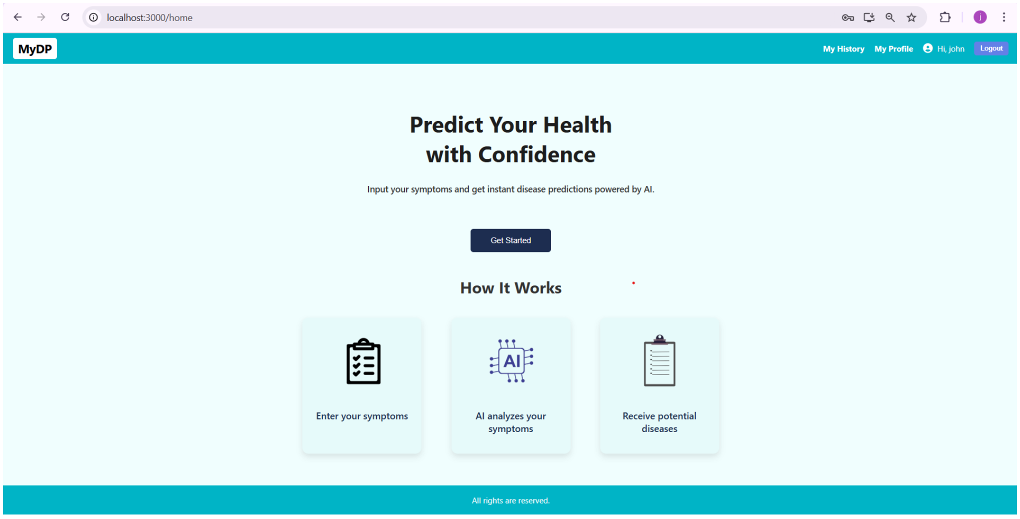


Figure 5.7: Actual Home Page

5.4.5 Select Input Method Page

The symptom input interface forms a key component of the system, allowing users to enter their health-related symptoms. To enhance usability, the page supports multiple input methods, including dropdown list and free-text entry.

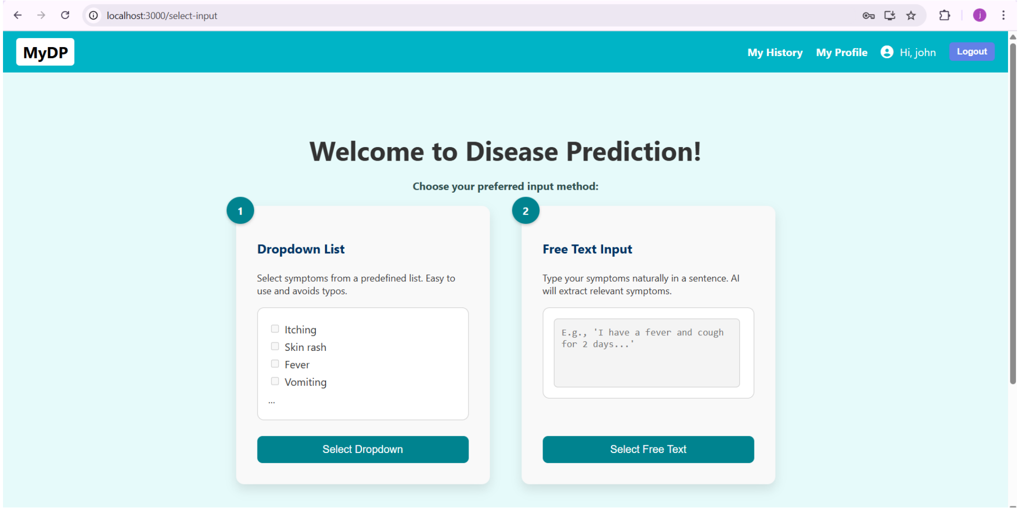


Figure 5.8: Actual Select Input Method Page

5.4.6 Dropdown List Input Symptoms Page

The Dropdown List Input Symptoms Page including dropdown lists, searchable fields, and checkbox selections, ensuring that users can effectively identify their symptoms. Users can see the selected symptoms before making predictions.

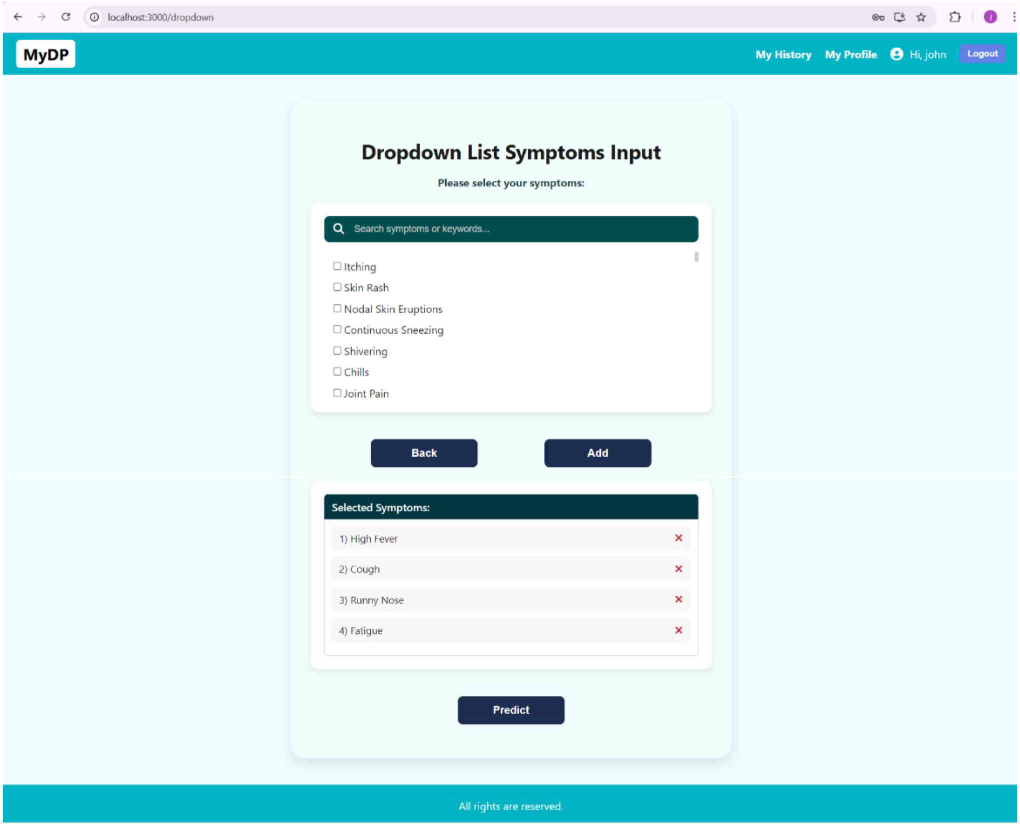


Figure 5.9: Actual Dropdown List Input Symptoms Page

5.4.7 Free Text Input Symptoms Page

The Free Text Input Symptoms Page allows users to enter free-text input symptoms, enabling greater flexibility in symptom submission. This design is intended to make the application more user-friendly, especially for the users who may not be familiar with the medical terminology or the exact symptom names used in the database.

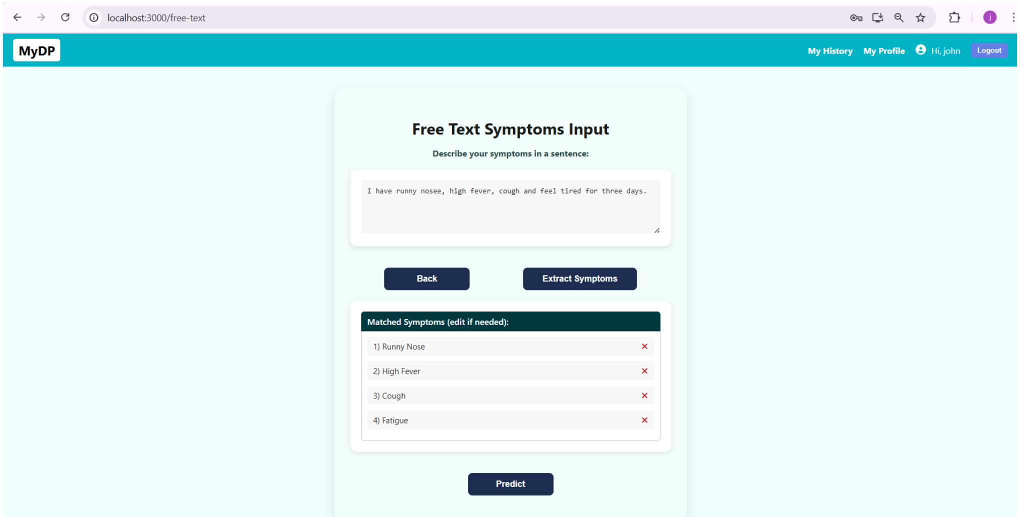


Figure 5.10: Actual Free Text Input Symptoms Page

5.4.8 Predicted Results Page

After submitting the input, the system generates prediction results and displays them on the results interface. This page highlights the predicted diseases in clear and concise format and shows the symptoms of users next to the potential diseases.

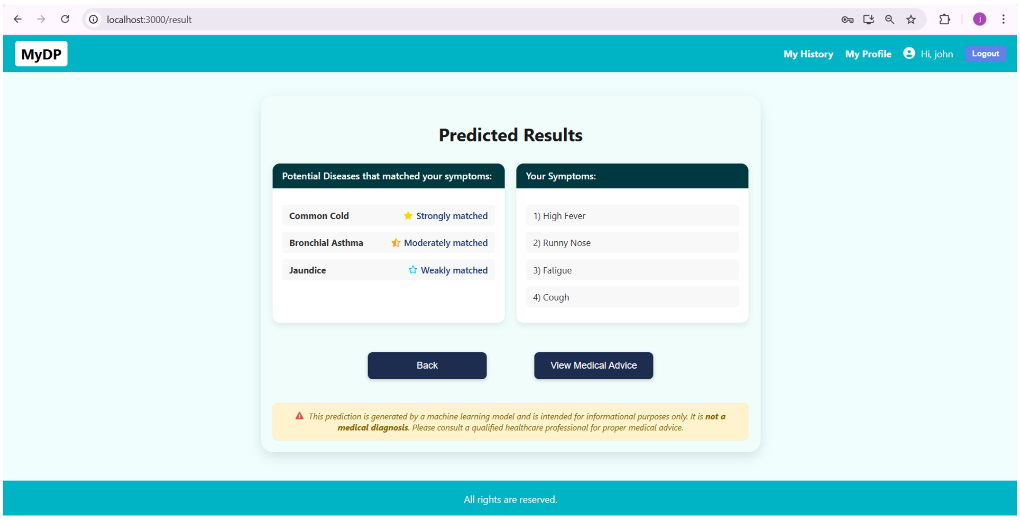


Figure 5.11: Actual Predicted Results Page

5.4.9 View Medical Advice Page

The View Medical Advice Page provides tailored description, lifestyle tips, prevention tips, and guidance on when to seek treatment for each potential disease. This can assist users in understanding potential next steps.

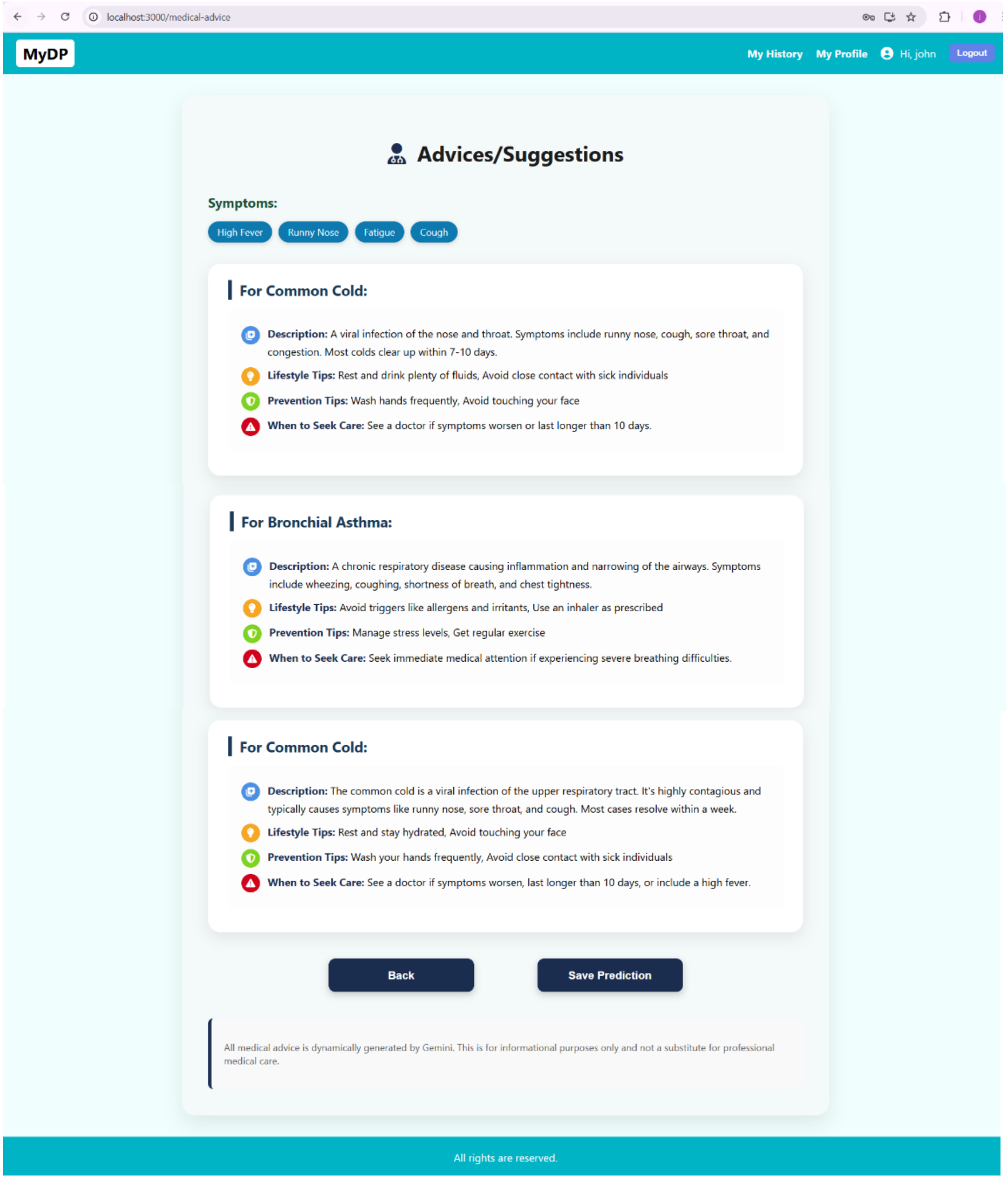


Figure 5.12: Actual View Medical Advice Page

5.4.10 History Page

The users can view records of the past prediction by navigating to the History Page. This feature displays previously entered symptoms, corresponding predictions results, and timestamps. Users may also delete the history records.

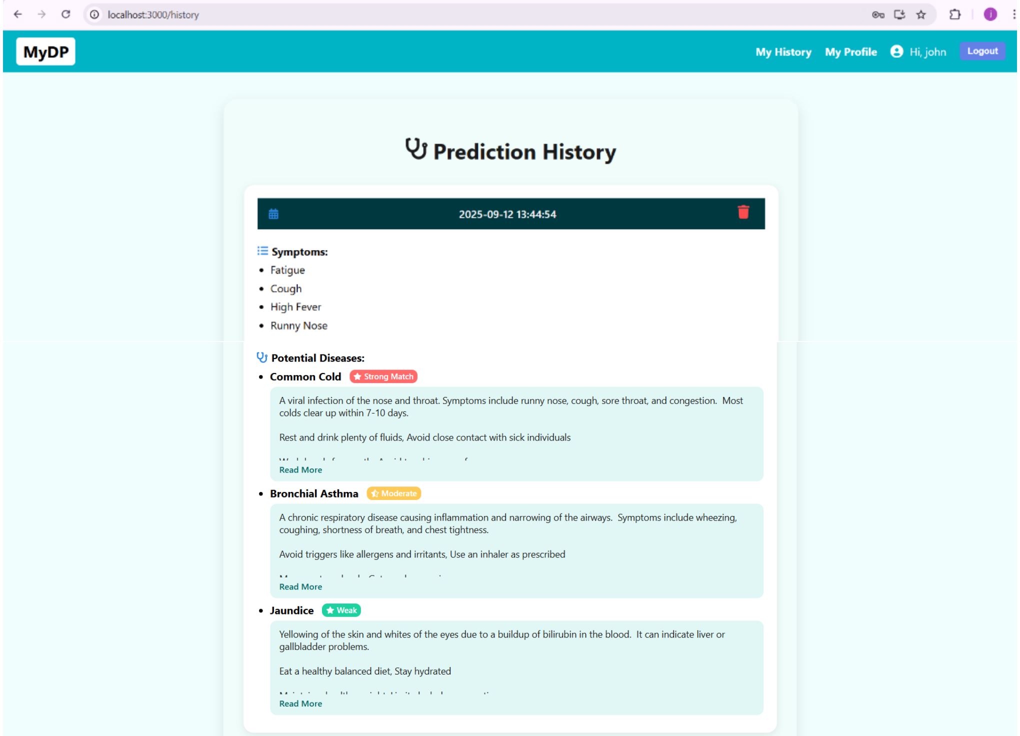


Figure 5.13: Actual History Page

5.4.11 Profile Page

The profile page allows users to view their personal details such as email address, date of birth and gender. The page also features an "Update Profile" button that directs users to the update profile page.

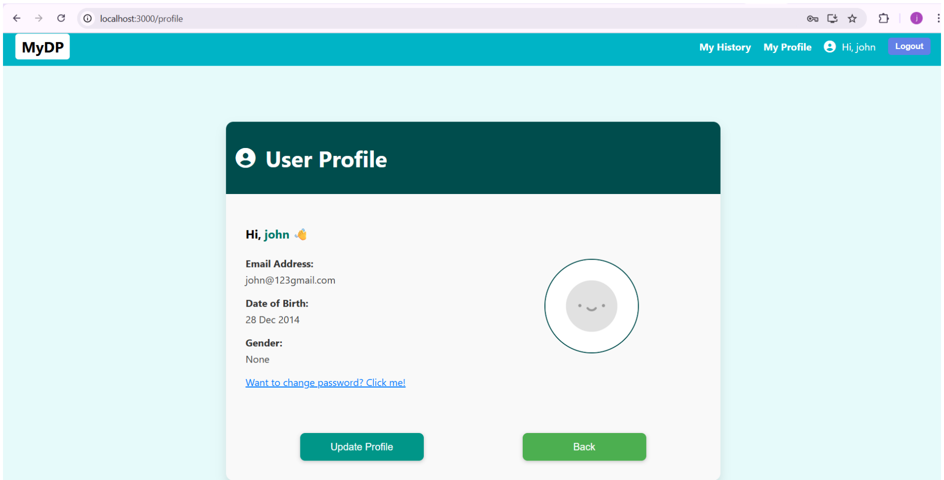


Figure 5.14: Actual Profile Page

5.4.12 Update Profile Page

The update profile page allows users to update personal information or change account credentials.

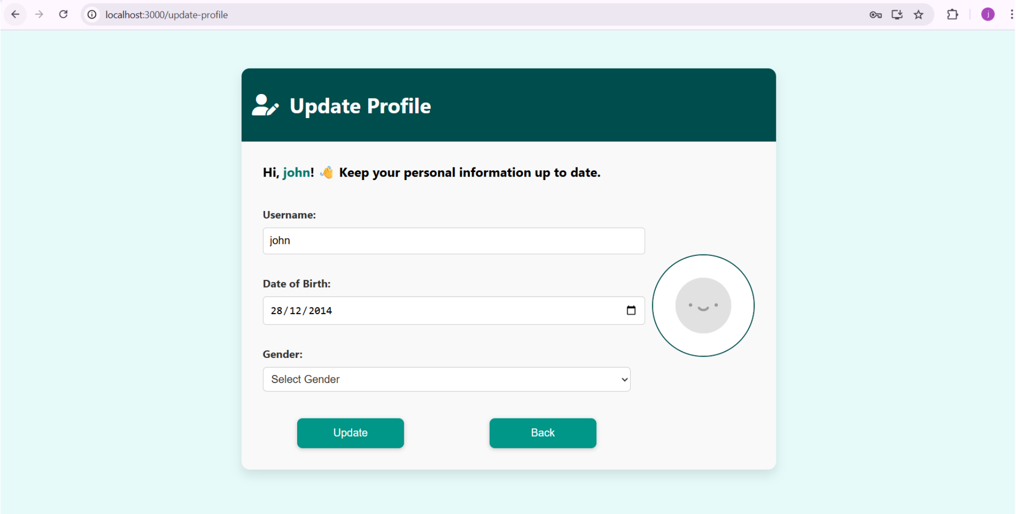


Figure 5.15: Actual Update Profile Page

5.4.13 Change Password Page

This page allows users to change their password.

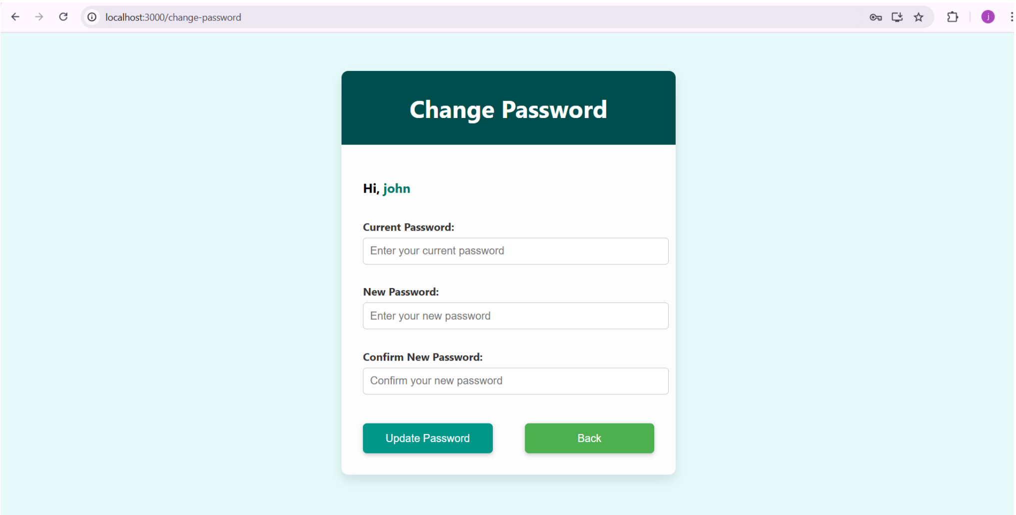


Figure 5.16: Actual Change Password Page

5.5 Prompt Design Study

The aim of this study is match to objective 3 of this project, which is to design and test different prompts for large language model (Google Gemini), evaluating their effectiveness in advice generation and validate the outputs against trusted medical sources. This study is to evaluate how different prompt designs influence the accuracy and reliability of automatically generated general medical advice and to validate the output results against trusted medical sources such as WHO, CDC and other.

5.5.1 Prompt Styles

There are three different prompt styles were designed and tested:

1. **Zero-shot Prompt**
2. **Role-based Prompt**
3. **Step by step reasoning Prompt**

5.5.1.1 Zero-shot Prompting

The zero-shot prompting are the straightforward and specific requests that clearly guide the model what to do or answer without the need for context and roles. This type of prompt is suitable for simple tasks where the user has a clear understanding of the output (Gadesha, 2025). It generates responses solely using the internal knowledge base of the model. The more precise the instructions, the greater the likelihood of achieving the desired outcome. It focuses on core action, making it an ideal choice for simple tasks such as generation or summarization (Jaiman, 2024). For example, a direct instruction prompt could be "Give general medical advice for a patient with diabetes." The strengths of this type of prompt are easy to design, enable to produces quick and concise responses. The weakness of it is the outputs may be incomplete, ambiguous or overly general since no additional guidance is given.

5.5.1.2 Role-based Prompting

A role-based prompt instructs the model to assume a specific professional or authoritative role before generating the response (GeeksforGeeks, 2025). This approach can influence the tone, styles and content of AI, making the output more pertinent, expert, and context aware. This facilitates the customization of responses to simulate specific viewpoint, ensuring greater contextually relevant and consistent. For example, the role-based prompt could be “You are a health assistant. Provide general advice for a patient with diabetes based on clinical guidelines.” This approach often produces more concise, structured, professional and authoritative outputs that align more closely with guideline-based practice. However, this type of prompt does not guarantee the factual correctness, the accuracy still depends on the model’s training data.

5.5.1.3 Chain-of-Thought (CoT) Prompting

The Chain-of-Though (CoT) prompting is a prompt engineering technique designed to guide the Large Language Models (LLMs) to generate explicit reasoning steps before arriving at a final answer. According to Gadesha, Kavlakoglu and Winland, this technique significantly enhances the quality of LLM outputs, particularly for the complex tasks involving multi-step reasoning. This does not provide the direct command but instead guides the model to “think out loud” by breaking down the problem into smaller logical steps (Gadesha, Kavlakoglu and Winland, 2025). For example, the chain-of-though prompting could be “Explain reasoning step by step about how a doctor would recommend on lifestyle advice for a patient with diabetes and then provide the final advice.” This approach enhances the completeness by making the model is less prone to skip the key steps. However, CoT prompts may generate the verbose and less concise outputs. The reasoning steps it generates may not always reflect the actual medical decision-making process.

5.5.2 Prompt Testing

The prompt testing was conducted using Google Gemini, where three different prompt styles (Zere-shot, Role-based, and Chain-of-Thought) were applied to identical medical queries. The generated outputs were collected and evaluated against the trusted medical sources. The evaluation was carried out using four criteria, which are accuracy, clarity, completeness, and consistence with established medical knowledge. To ensure consistency and comparability, all generated response were produced using Google Gemini 2.5 Flash model and limited to a maximum of 100 words.

5.5.2.1 Prompt Testing on Diabetes

The prompt test is focused on the diabetes disease. There are three different prompting styles were tested to generate the general medical advice for a patient with diabetes.

5.5.2.1.1 Zero-shot Prompting

Q: Give general medical advice for a patient with diabetes, not exceeding 100 words.

A: Managing diabetes effectively is key to a healthy life. Regularly monitor your blood sugar levels as advised by your doctor. Follow a balanced diet rich in whole grains, lean proteins, and vegetables, while limiting sugary drinks and processed foods. Incorporate regular physical activity into your routine, such as walking or swimming. Take your medications as prescribed and attend all scheduled medical appointments. Stay hydrated and manage stress. If you notice unusual symptoms, contact your healthcare provider immediately.

Analysis of Zero-shot prompt

This output is aligned with the Cleveland Clinic (Cleveland Clinic, 2023) and National Library of Medicine (Reynolds and Mitri, 2024) guidelines, which recommend regular blood sugar monitoring, balanced diet and limiting processed foods and sugary drinks. It also consistent with Mayo Clinic advice on physical activities and stress management (Mayo Clinic Staff, 2025). The response also includes a red-flag warning such as “contact healthcare provider if unusual symptoms”, which is supported by NHS guidelines (NHS Choices, 2020). Overall, the answer provided is accurate and comprehensive. However, it does not mention a specific exercise target such as 150 minutes per week, making it slightly less precise.

5.5.2.1.2 Role-based Prompting

Q: You are a health assistant. Provide general advice for a patient with diabetes based on clinical guidelines, not exceeding 100 words.

A: For diabetes management, prioritize a balanced diet rich in whole grains, fruits, and vegetables while limiting sugary foods and saturated fats. Engage in regular physical activity, aiming for at least 150 minutes of moderate exercise weekly. Monitor blood glucose levels as advised by your doctor and take medications as prescribed. Stay hydrated, manage stress, and get adequate sleep. Regular check-ups are crucial for monitoring complications. Always consult your healthcare provider for personalized advice.

Analysis of Role-based Prompt

This response aligns closely with the guidelines from Medline Plus (MedlinePlus, 2019) and Mayo Clinic (Mayo Clinic, 2024), which recommend a healthy, balanced diets and engaging in at least 150 minutes of moderate exercise per week. It also incorporates monitoring the blood sugar levels, hydration and managing stress, also reflecting Mayo Clinic advice. The advice of "regular check-ups" is consistent with World Health Organization (WHO, 2024) guidelines for long-term monitoring. Overall, this output is accurate and reliable, meets guideline requirements, and more precise than the Zero-shot response.

5.5.2.1.3 Chain-of-Thought (CoT) Prompting

Q: Explain reasoning step by step about how a doctor would recommend on lifestyle advice for a patient with diabetes and then provide the final advice, not exceeding 100 words.

A: A doctor's recommendation for a patient with diabetes involves a multi-faceted approach.

1. **Assessment:** The doctor first assesses the patient's current lifestyle, including diet, physical activity level, smoking habits, alcohol

consumption, and stress levels. They also consider the patient's age, weight, and any coexisting medical conditions.

2. **Education:** The doctor educates the patient on the importance of managing blood sugar levels and the long-term health risks of uncontrolled diabetes.

3. **Personalization:** The doctor then tailors the advice to the patient's specific needs and preferences. For example, a busy professional might receive advice on meal prep, while a less active person might get guidance on incorporating short walks into their daily routine.

4. **Goal Setting:** The doctor works with the patient to set realistic and achievable goals, such as losing a specific amount of weight or aiming for a certain number of steps per day.

5. **Follow-up:** The doctor schedules follow-up appointments to monitor progress, address challenges, and make adjustments to the plan as needed.

Final Advice: Manage diabetes through a balanced diet of whole foods, regular exercise (aim for at least 150 minutes of moderate activity weekly), and weight management. Monitor blood sugar, take medications as prescribed, and reduce stress. Limit processed foods, sugary drinks, and alcohol, and avoid smoking.

Analysis of CoT Prompt

This answer provides achieves transparent reasoning by simulating a doctor’s decision-making process. The final advice matches Mayo Clinic (Mayo clinic, 2025) guidelines, including the weekly 150 minutes exercise target, dietary control, and stress management. It also includes red-flag symptoms for urgent care, consistent with NHS guidance. While the content is accurate and comprehensive, including reasoning steps make it more verbose, which slightly reduces clarity for patient-facing advice. Additionally, compared to zero-shot and role-based responses, this response takes longer to generate.

5.5.2.1.4 Summary of Diabetes Results

Table 5.8: Diabetes Results on Prompt Testing

Prompt Style	Zero-shot	Role-based	Chain-of-Thought
Accuracy	Accurate, but missed specific exercise target	Accurate and aligns with guidelines	Accurate and guideline-consistent
Clarity	Clear and easy to understand	Very clear and concise	Less clear due to added reasoning steps
Completeness	Mostly complete, but lacking in detail	Comprehensive, covers all key aspects	Comprehensive, includes reasoning and final advice
Consistency	Consistent with trusted sources	Strongly consistent with trusted sources	Consistent, but reasoning makes it less concise
Response Time	Fast response, very quick generation	Slightly slower, as role framing adds context	Slowest, as reasoning steps make the response longer

All three outputs cover the essential high-level point. Role-based and Chain-of-Thought provide slightly more clearer targets. Responses generated by each prompt style are validated against authoritative recommendations on balanced diet, regular physical activity, blood glucose monitoring and follow-up, medication adherence, and when to seek medical care. These guidelines align with WHO, CDC, Mayo Clinic and NHS guidelines. Role-based prompting generates concise, guideline-aligned advice, including explicit weekly exercise targets of approximately 150 minutes. Chain-of-Thought prompting provided richer reasoning and personalized content but are longer in length. Zero-shot generation provides acceptable high-level advice. Overall, the Role-based prompting proves most effective at generating concise, guideline-consistent advice for diabetes management.

5.5.2.2 Prompt Testing on Hypertension

The prompt test is focussed on the hypertension disease. There are three different prompting styles to generate the general medical advice for a patient with hypertension.

5.5.2.2.1 Zero-shot Prompting

Q: Give general medical advice for a patient with hypertension, not exceeding 100 words.

A: Managing hypertension, or high blood pressure, is crucial for your health. A low-sodium diet is key; avoid processed foods and added salt. Incorporate regular aerobic exercise, like brisk walking, for at least 30 minutes most days of the week. Maintain a healthy weight, as even a small reduction can significantly lower blood pressure. Adhere strictly to your medication schedule as prescribed by your doctor and attend all follow-up appointments. Limit alcohol consumption and quit smoking. Monitor your blood pressure at home and report any concerning readings to your healthcare provider.

Analysis of Zero-shot Prompt

This response covers key advice consistent with the WHO (World Health Organization, 2023) guidance, including healthy diet, reduced salt, exercise at least 30 minutes most days, medication adherence, and stress management. For the limited alcohol consumption and quit smoking are also aligns with Hello Doktor (2017) health and wellness website. It also incorporates monitoring blood pressure, reflecting to Centers for Disease Control and Prevention | CDC advice (CDC, 2024). However, it does not specify quantitative targets for salt reduction, which are detailed in WHO (World Health Organization, 2025) guidelines such as salt <5 g per day.

5.5.2.2.2 Role-based Prompting

Q: You are a health assistant. Provide general advice for a patient with hypertension based on clinical guidelines, not exceeding 100 words.

A: For managing hypertension, focus on a healthy lifestyle. Adopt a low-sodium diet rich in fruits, vegetables, and whole grains, and limit processed foods. Engage in regular physical activity, aiming for at least 150 minutes of moderate exercise per week. Maintain a healthy weight, limit alcohol intake, and quit smoking. Monitor your blood pressure regularly and take all prescribed medications as directed. Consult your healthcare provider for personalized guidance and treatment plans.

Analysis of Role-based Prompt

The Role-based output is highly guideline-oriented. It explicitly heart-healthy dietary principles and specifies a weekly activity target (≥150 minutes), includes medication adherence, and advises regular blood-pressure monitoring and urgent care for severe symptoms. This matches WHO (World Health Organization, 2021) recommendations and is especially useful for clear patient instructions because of its specificity and brevity. Role-based prompting consistently generates answers that comply with guidelines and well-suited for integration into patient educational material.

5.5.2.2.3 Chain-of-Thought (CoT) Prompting

Q: Explain reasoning step by step about how a doctor would recommend on lifestyle advice for a patient with hypertension and then provide the final advice, not exceeding 100 words.

A: A doctor's recommendation for hypertension management involves a thorough, personalized approach.

Reasoning Steps

1. **Diagnosis & Risk Assessment:** The doctor first confirms the diagnosis of hypertension through multiple blood pressure (BP) readings. They assess the patient's overall cardiovascular risk, considering factors like age, family history, and other conditions such as diabetes or high cholesterol. This determines the urgency and intensity of lifestyle changes.

2. **Educate & Motivate:** The doctor explains to the patient what hypertension is, why it's a "silent killer" with often no symptoms, and how lifestyle changes can directly lower BP and reduce the risk of serious complications like stroke or heart attack. They emphasize that these changes are often the first line of treatment.

3. **Tailor the Plan:** The doctor creates a personalized plan based on the patient's current habits. For example, they might recommend the DASH diet for someone with high sodium intake or specific physical activities for a sedentary patient. The goals are made realistic and achievable to ensure long-term adherence.

4. **Monitor & Adjust:** The doctor schedules follow-up appointments to track the patient's BP, discuss progress, and address any challenges. They adjust the plan, which may include adding medication if lifestyle changes alone are not sufficient to reach target BP goals.

Final Advice

Adopt a **DASH-style diet** rich in fruits, vegetables, and low-fat dairy, while reducing sodium intake. Engage in **regular aerobic exercise**, aiming for 150 minutes weekly. Maintain a healthy weight, limit alcohol, and quit smoking. These changes significantly lower blood pressure and reduce the risk of heart disease and stroke.

Analysis of CoT Prompt

The CoT response includes step-by-step clinical recommendations, followed by a final advice block that follow guidelines from Medline Plus (Berman, 2022), Mayo Clinic (Mayo clinic, 2025) and the Hello Doktor (2017). These include the DASH diet, 150 minutes of exercise per week, limited alcohol and stop smoking. The CoT format enhances interpretability and shows the clinical logic behind recommendations, which is valuable for auditing review. However, the additional reasoning text reduces the conciseness of information delivery, making it less suitable for direct patient use and may need to be trimmed if the output is shown to patients.

5.5.2.2.4 Summary of Hypertension Results

Table 5.9: Hypertension Results on Prompt Testing

Prompt Style	Zero-shot	Role-based	Chain-of-Thought
Accuracy	Accurate, advice matches activity targets	Highly accurate, explicitly guideline-aligned	Accurate, includes DASH and exercise recommendations
Clarity	Clear and easy to understand	Very clear and structured	Less clear due to lengthy reasoning steps
Completeness	Complete, covers diet, exercise, medication and lifestyle	Comprehensive, covers key recommendations	Complete, includes reasoning and final advice
Consistency	Consistent with trusted sources	Consistent with trusted sources	Consistent, but verbose compared to guideline wording
Response Time	Fast, almost immediate response	Slightly slower due to role framing adds context	Slowest, as reasoning steps extends generation time

Same as summary of Diabetes results, all three outputs cover the essential high-level point. Role-based and Chain-of-Thought provide slightly more clearer targets. Responses generated by each prompt style are validated against authoritative recommendations on balanced diet, regular physical activity, blood pressure monitoring and follow-up, medication adherence, and when to seek medical care. These guidelines align with WHO, CDC, Mayo Clinic and NHS guidelines. Chain-of-Thought prompting provided richer reasoning and personalized content but are longer in length. Zero-shot generation provides acceptable high-level advice. Overall, the Role-based prompting proves most effective at generating concise, guideline-consistent advice for hypertension management.

5.5.3 Comparison and Discussion

A comparative analysis of three prompting styles, which are Zero-shot, Role-based, and Chain-of-Thought for both diabetes and hypertension revealed significant clear differences in accuracy, clarity, completeness, consistency and response efficiency.

5.5.3.1 Zero-shot Prompting

Zero-shot prompts generated reasonably accurate and clear suggestions in both scenarios. However, such suggestions often lack completeness, frequently omitting specific exercise goals or guideline-based recommendations. The primary advantage of this prompt lies in its ability to generate responses almost instantly, but at the cost of sacrificing depth.

5.5.3.2 Role-based Prompting

Role-based prompts consistently deliver the most reliable and guideline-compliant responses. By positioning the model as a "health assistant," its recommendations become more structured, aligned with patient needs, and consistent with the trust authoritative medical institutions such as the WHO or CDC. This approach strikes a balance between accuracy, clarity, and completeness while maintaining reasonable response times.

5.5.3.3 Chain-of-Thought (CoT) Prompting

Cot prompted responses are comprehensive and contextually rich, revealing the reasoning behind medical advice to enhance transparency. However, this advantage comes at the cost of reduced clarity due to verbosity and logical complexity and slower response times. While the advice is accurate and complete, the length reasoning process may be difficult for patients seeking quick, actionable guidance to digest.

These results indicate that Role-based prompting strategies represent the most effective approach for generating general medical advice using large language models. This strategy strikes a balance between accuracy, clarity, completeness, and efficiency, making it suitable for real-world health consultation applications. While CoT prompts hold value in enhancing transparency and achieving high-level reasoning, but it more suitable for backend verification on professional user scenarios rather than direct patient communication. Zero-shot prompting styles offer speed advantages but lack the necessary reliability and specific guidance.

5.5.4 Summary

This short study evaluated three prompting styles, which are Zero-shot, Role-based, and Chain-of-Thought for generate medical advice for diabetes and hypertension using large language models (Google Gemini). The outputs were compared against trusted medical resources based on 5 metrics including accuracy, clarity, completeness, consistency, and response efficiency.

The results showed that Role-based prompting produced the most well-balanced and guideline-consistent outputs with accuracy, clarity, and completeness as well as maintaining reasonable response time. Zero-shot prompting is fast and straightforward, but lacking in depth and completeness, hence comparatively less reliable. Chain-of-Thought prompting produces comprehensive and transparent reasoning steps, but its length expression lowers clarity and increased response time, potentially limiting its suitability for patient-facing contexts.

Overall, role-based prompting strategies represent the most effective approach for generating accurate and accessible health advice. This method ensures precision and consistency with clinical guidelines while providing clear and concise guidelines for patient use. These findings confirm that well-designed prompts are crucial for optimizing the usability of large language models in medical applications, aligning with the project goal of designing and testing effective prompting strategies.

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

This chapter details the implementation of the Disease Prediction Web Application using Machine Learning, outlining the development process for the machine learning model development, frontend development, backend development, large language model integration and database configuration. The system was implemented using a React frontend, a Flask backend, a MySQL relational database, and a scikit-learn machine learning model.

6.2 Machine Learning Model Development

The Machine Learning Model was implemented using scikit-learn and trained on the Symptom-Disease Prediction Dataset (SDPD) published by Jay Tucker in 2024. The scikit-learn library offers a wide range of classifications algorithms. The development process includes dataset preprocessing, model training and evaluation. There are three models were selected for comparison, with the highest-performing model ultimately integrated into the web application.

6.2.1 Data Preprocessing

The dataset contains symptom-disease mappings. Each record includes a set of symptoms as input features and the corresponding disease as the target label. The dataset used in this project is structured in a binary format, where each symptom is represented by 0 or 1. 0 indicates absence, while 1 indicates presence. The data preprocessing steps were designed to handle missing value, categorical label encoding for the target, remove duplicates data, and split the dataset to training set, validation set and testing set.

6.2.1.1 Handling Missing Value

Missing or incomplete data may degrade model performance and introduce bias. Therefore, rows containing missing values (NaN) will be removed. The dataset has completed missing value checks, since no missing values were detected, no imputation or deletion operations are required. This ensures the integrity of the dataset while preserving its original distribution. Figure 6.2 shows a code snippet for handling missing value, where missing values are treated as zero.

```
df.isnull().sum()
print(f"Missing values: {df.isnull().sum().sum()}")
```
```
Missing values: 0
```

Figure 6.1: Code Snippet for Handling Missing Value

6.2.1.2 Categorical Label Encoding

The target variable (prognosis) in the dataset consists of disease names, which are categorical labels such as Diabetes or Fungal Infection. Since most machine learning algorithms cannot directly process string labels, scikit-learn internally converts these disease names into numerical representations during model training. This process ensures classification models can effectively recognize and distinguish between different disease categories while preserving the original dataset structure.

6.2.1.3 Duplicate Removal

Duplicate records may introduce bias into the training process and potentially reduce processing speed, particularly when handling relatively small datasets. To address this issue, the dataset underwent duplicate scanning, and all duplicate rows were removed. This ensures each observation contributes equally to the training process, preventing overfitting to repeated samples. Figure 6.1 shows the code snippet of remove duplicate rows in dataset.

```
df = pd.read_csv("symbipredict_2022.csv")
df = df.drop_duplicates().reset_index(drop=True)
```

Figure 6.2: Code Snippet for Remove Duplicate Rows in Dataset

6.2.1.4 Dataset Splitting

After data cleaning, the dataset was divided into three subsets, which are training set, validation set and testing set. Following the standard machine learning practice, the splitting ratio was set to 70% for training, 15% for testing and 15% for validation. To ensure the robustness of the evaluation, at least one sample from each disease category is randomly selected for inclusion in both test and validation sets. The remaining data was allocated proportionally to maintain the target distribution across all data splits. Figure 6.3 shows the code snippet for data splitting. The three subsets will be saved in the project folder, which are Training.csv, Testing.csv, and Validation.csv. All the subsets are in CSV format and contain symptom features along with their corresponding target labels.

```
target_col = "prognosis"
total_len = len(df)

test_size = round(0.15 * total_len)
val_size = round(0.15 * total_len)
train_size = total_len - test_size - val_size

def select_random_per_class(df, target, base_seed=100):
    return df.groupby(target, group_keys=False).apply(
        lambda x: x.sample(1, random_state=(base_seed + hash(x[target].iloc[0]) % 1000))
    )

test_holdout = select_random_per_class(df, target_col, base_seed=100)
df_remaining = df.drop(test_holdout.index)
val_holdout = select_random_per_class(df_remaining, target_col, base_seed=200)
df_remaining = df_remaining.drop(val_holdout.index)
```

Figure 6.3: Code Snippet for Data Splitting

6.2.2 Model Training

Three models were selected for experimentation based on their suitability for classification tasks, which are Random Forest, Decision Tree and Support Vector Machine. The model training process involved training three machine learning models to select the best-performing model for the disease prediction task. Figure 6.4 shows the default version of three machine learning models architecture.

```
models = {
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "SVM": SVC()
}
```

Figure 6.4: Model Architecture

6.2.3 Model Optimization

The goal of model optimization is to determine the optimal combination of hyperparameters that enhance model performance. This can prevent issues such as overfitting or underfitting, improved accuracy, and better generalization. The hyperparameter tuning was conducted on three selected algorithms. The tuning process employed a manual grid search methos, systematically evaluating candidate parameter values on the validation dataset. Model performance was assessed using accuracy, precision, recall and F1 score.

1. **Random Forest Classifier:** The default RandomForestClassifier() uses 100 trees (n_estimators=100) with no depth restriction (max_depth=None). While this offers flexibility, it may lead to unnecessary complexity.
   Tuning Parameters:
   - max_depth: [5, 9, 10, 15],
   - n_estimators: [50, 100]

2. **Decision Tree Classifier:** By default, DecisionTreeClassifier() grows decision tress until all leaf nodes are pure classes.
   Tuning Parameters:
   - criterion: ['gini', 'entropy'],
   - max_depth: [None, 5, 10]

3. **Support Vector Machine (SVM):** The default SVC() uses a radial basis function (RBF) kernel with C=1.0 and gammar='scale'. This baseline model may fail to capture complex patterns in high-dimensional data.
   Tuning Parameters:
   - C: [0.1, 1, 10],
   - gamma: ['scale', 0.01, 0.001],
   - kernel: ['rbf']

Each model was evaluated on the validation set using multiple performance metrics, including performance metrics, including accuracy, precision, recall, F1-score. The hyperparameters yielding the highest validation accuracy were ultimately selected for retraining on the full training set and test with testing set. Figure 6.5 shows the hyperparameter tuning for three models.

```
rf_param_grid = {
    "n_estimators": [50, 100],
    "max_depth": [5, 9, 10, 15]
}

svm_param_grid = {
    "C": [0.1, 1, 10],
    "gamma": ['scale', 0.01, 0.001],
    "kernel": ['rbf']
}

dt_param_grid = {
    "criterion": ['gini', 'entropy'],
    "max_depth": [None, 5, 10]
}
```

Figure 6.5: Hyperparameter Tuning for three models

After evaluation on the validation set, the optimal hyperparameter for each model were determined as follows:

- Random Forest: n_estimators = 100, max_depth = 9
- Decision Tree: criterion = gini, max_depth = None
- Support Vector Machine: C = 10, gamma = 0.001, kernel =rbf

These optimizes configurations outperformed the baseline model on the validation set and were therefore selected for subsequent testing and final evaluation.

6.2.3.1 Model Optimization Results

This section presents the model optimization results. Figure 6.6 to 6.10 present detailed results of hyperparameter tuning on the validation dataset. These figures illustrate the performance of each candidate configuration across multiple evaluation metrics, including accuracy, precision, recall and F1-score.

Random Forest

The optimization results indicate that certain configurations of Random Forest achieved perfect validation score, with the accuracy, precision, recall and F1-scores all reaching 1.000. While such results appear outstanding at first glance but raise concerns of overfitting. Overfitting occurs when the model memorizes the training and validation sets rather than learning generalized patterns, thereby weakening its ability to handle unseen data. The configuration with n_estimators = 50/100 and max_depth = 15 consistently produced perfect scores, which were treated as overfit models. In contrast, more balanced configurations are n_estimators = 100 and max_depth = 9, which contribute the accuracy of 97.83%. The Figure 6.6 shows the Random Forest optimization results. Figure 6.7 shows the learning curve of the Random Forest, indicate that the max_depth with 9 has the highest accuracy without overfitting.

```
Tuning Random Forest...
Params: {'n_estimators': 50, 'max_depth': 5, 'random_state': 42} | Val Accuracy: 0.8043 | Precision: 0.7297 | Recall: 0.8049 | F1: 0.7528
Params: {'n_estimators': 50, 'max_depth': 9, 'random_state': 42} | Val Accuracy: 0.9565 | Precision: 0.9512 | Recall: 0.9634 | F1: 0.9512
Params: {'n_estimators': 50, 'max_depth': 10, 'random_state': 42} | Val Accuracy: 0.9565 | Precision: 0.9593 | Recall: 0.9756 | F1: 0.9634
Params: {'n_estimators': 50, 'max_depth': 15, 'random_state': 42} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Params: {'n_estimators': 100, 'max_depth': 5, 'random_state': 42} | Val Accuracy: 0.8261 | Precision: 0.7659 | Recall: 0.8293 | F1: 0.7855
Params: {'n_estimators': 100, 'max_depth': 9, 'random_state': 42} | Val Accuracy: 0.9783 | Precision: 0.9634 | Recall: 0.9756 | F1: 0.9675
Params: {'n_estimators': 100, 'max_depth': 10, 'random_state': 42} | Val Accuracy: 0.9565 | Precision: 0.9593 | Recall: 0.9756 | F1: 0.9634
Params: {'n_estimators': 100, 'max_depth': 15, 'random_state': 42} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
```

Figure 6.6: Random Forest Optimization Results

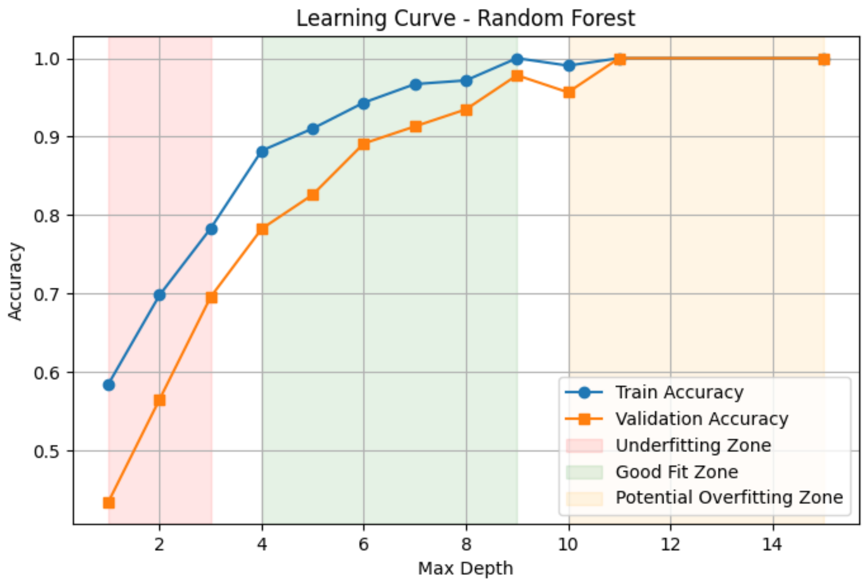


Figure 6.7: Learning Curve of Random Forest

Support Vector Machine

Same as Random Forest, the optimization results indicate that certain configurations achieved perfect validation score, with the accuracy, precision, recall and F1-scores all reaching 1.000. Overfitting may occur also in the Support Vector Machine. The configuration with C = 1 or C =10 and gamma set to ‘scale’ also result in perfect validation performance, suggesting potential overfitting. In contrast, more balanced configurations are 'C': 10, and 'gamma':

0.001, which contribute the accuracy of 93.4%. The Figure 6.8 shows the Support Vector Machine optimization results. Figure 6.9 shows the heatmap diagram of SVM to compare different values of C and gamma to visualize the accuracy of grid.

```
Tuning SVM...
Params: {'C': 0.1, 'gamma': 'scale', 'kernel': 'rbf'} | Val Accuracy: 0.1087 | Precision: 0.0604 | Recall: 0.1220 | F1: 0.0673
Params: {'C': 0.1, 'gamma': 0.01, 'kernel': 'rbf'} | Val Accuracy: 0.1522 | Precision: 0.1085 | Recall: 0.1707 | F1: 0.1150
Params: {'C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'} | Val Accuracy: 0.0652 | Precision: 0.0331 | Recall: 0.0732 | F1: 0.0377
Params: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Params: {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Params: {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'} | Val Accuracy: 0.6739 | Precision: 0.6308 | Recall: 0.6829 | F1: 0.6407
Params: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Params: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'} | Val Accuracy: 1.0000 | Precision: 1.0000 | Recall: 1.0000 | F1: 1.0000
Params: {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'} | Val Accuracy: 0.9348 | Precision: 0.9085 | Recall: 0.9268 | F1: 0.9122
```

Figure 6.8: Support Vector Machine Optimization Results

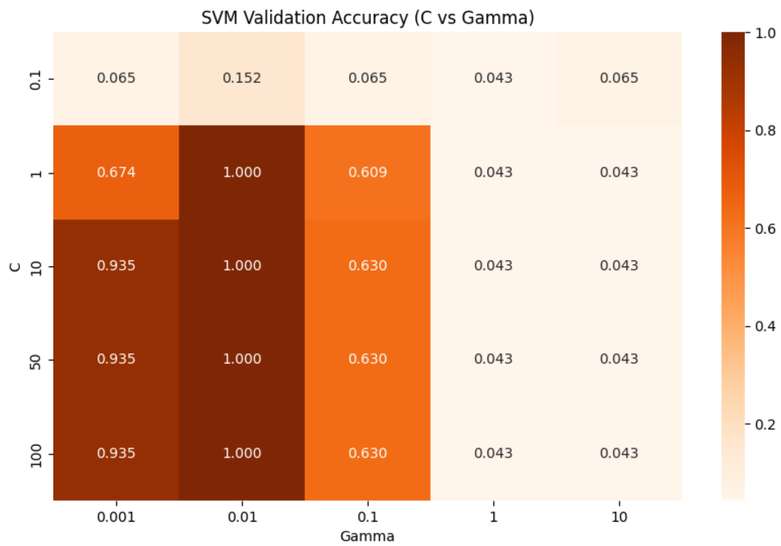


Figure 6.9: Heatmap diagram of SVM

Decision Tree

The tuning results of the Decision Tree classifier indicate that clear differences in performance based on the selected criterion and maximum depth. When trained with the default configuration (criterion= 'gini', max_depth=None), the model achieved the highest validation performance with an accuracy of 71.74%. This demonstrates that an unrestricted depth enables the decision tree to capture the underlying patterns within the dataset effectively. In contrast, limiting the maximum depth to 5 or 10 cause a sharp decline in performance. Such results indicate that underfitting and the model is overly simplified and fails to adequately capture the complexity of the data. The Figure 6.10 shows the Decision Tree optimization results.

```
Tuning Decision Tree...
Params: {'criterion': 'gini', 'max_depth': None, 'random_state': 42} | Val Accuracy: 0.7174 | Precision: 0.6900 | Recall: 0.7195 | F1: 0.6875
Params: {'criterion': 'gini', 'max_depth': 5, 'random_state': 42} | Val Accuracy: 0.1087 | Precision: 0.0981 | Recall: 0.1220 | F1: 0.0987
Params: {'criterion': 'gini', 'max_depth': 10, 'random_state': 42} | Val Accuracy: 0.2391 | Precision: 0.2208 | Recall: 0.2439 | F1: 0.2220
Params: {'criterion': 'entropy', 'max_depth': None, 'random_state': 42} | Val Accuracy: 0.5652 | Precision: 0.4927 | Recall: 0.5610 | F1: 0.4988
Params: {'criterion': 'entropy', 'max_depth': 5, 'random_state': 42} | Val Accuracy: 0.3913 | Precision: 0.3354 | Recall: 0.4268 | F1: 0.3474
Params: {'criterion': 'entropy', 'max_depth': 10, 'random_state': 42} | Val Accuracy: 0.5217 | Precision: 0.4512 | Recall: 0.5488 | F1: 0.4704
```

Figure 6.10: Decision Tree Optimization Results

6.2.4 Model Evaluation

The optimized models were subsequently tested on the independent testing set to evaluate the model's performance. For each algorithm, predictions were compared with against actual labels, and metrics such as accuracy, precision, recall and F1-score were calculated. Evaluation is crucial for determining whether a model can effectively generalize its training results to unseen data, rather than merely performing well on the training and validation set.

The optimal hyperparameter were determined for all three models, the models were retained and evaluated on the independent testing dataset. This dataset was not used during the training or validation process, ensuring that the evaluation results accurately reflect the generalization capabilities of each model. The results of the evaluation are presented in the Figure 6.11. The Figure 6.12 shows the bar chart of the model performance on testing set.

--- Test Set Metrics ---

	Accuracy	Precision	Recall	F1-score
Random Forest	0.978261	0.963415	0.975610	0.967480
Decision Tree	0.652174	0.583740	0.634146	0.595492
SVM	0.956522	0.943089	0.951220	0.946341

Figure 6.11: Testing set results of 3 models

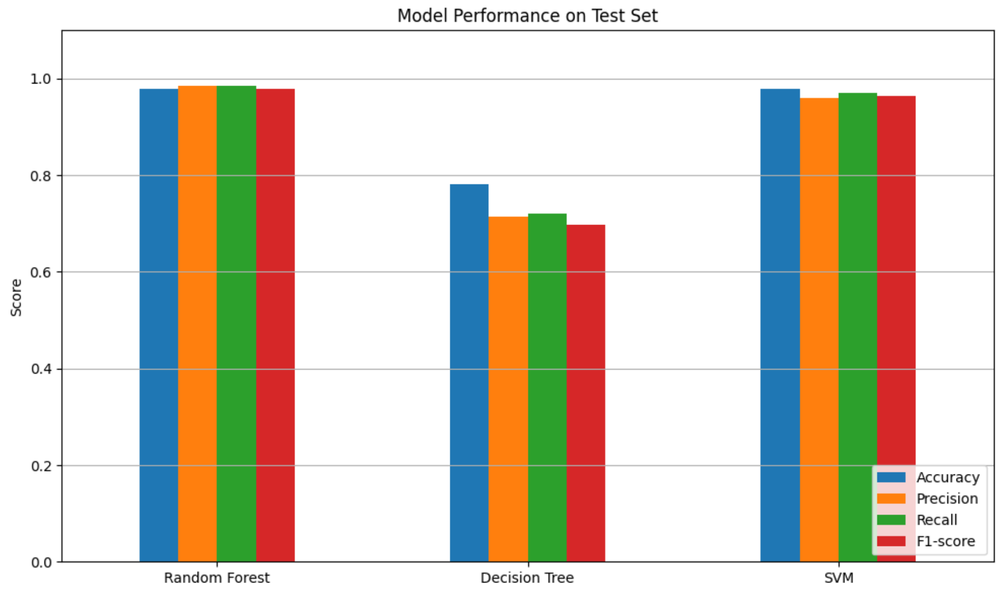


Figure 6.12: Model Performance on Test Set

The results show that the Random Forest (RF) model outperformed the other two algorithms across all evaluation metrics. With an accuracy rate of 97.8% and balanced performance in precision, recall and F1-score, the Random Forest demonstrated robust predictive capability and stability, making it the most suitable for integrate in the system. In contrast, the Support Vector Machine (SVM) also delivered robust performance with an accuracy rate of 95.7%. Although slightly inferior to Random Forest across all metrics, the SVM still demonstrated high predictive capability and strong generalization ability on unseen data. The Decision Tree (DT) achieved only 65.2% accuracy on the testing set, showing a significant gap compared to both Random Forest and Support Vector Machine.

Comparative analysis indicates that while both Random Forest and Support Vector Machine achieved highly reliable results, RF consistently outperformed SVM across all metrics. Although DT serve as useful baseline models, the predictive capabilities remain insufficient. Consequently, the RF model was ultimately selected as the ensemble prediction model for the disease prediction web application, as it achieves the optimal balance between accuracy, precision, recall and F1 score. The RF model is saved as "random_forest_model.pkl" in the backend.

To ensure the correctness and reliability of the Random Forest model, a practical validation approach was conducted in which predefined sets of symptoms were submitted to the trained model. The top 3 predicted diseases were then compared against conditions identified from trusted medical resources such as Mayo Clinic, MedlinePlus and WHO. This approach aims to validate whether predictions generated by machine learning models were medically validity, although the system is not intended to replace professional diagnosis.

The correctness of predictions was classified into three categories:

- **Exact Match:** The expected disease was among the system's top 3 predictions.
- **Partial Match:** The predicted disease was medically related but not the primary expected disease.
- **No Match:** The predicted diseases did not align with any expected conditions.

Table 6.1: Prediction Verification Results

Symptom Set	**Expected Disease(s)**	**System Predictions (Top 3)**	**Match Type**
High fever, Chills, Sweating	Malaria (Centres for Disease Control)	Malaria, Typhoid, Heart Attack	Exact
Polyuria, Excessive hunger, Weight loss	Diabetes (Mayo Clinic)	Diabetes, Jaundice, Hyperthyroidism	Exact
Chest pain, Breathlessness, Sweating, Palpitations	Heart Attack (Mayo Clinic)	Hypoglycemia, Heart Attack, Malaria	Partial
Red spots over body, Itching, Pus filled pimples	Chickenpox (Cleveland Clinic)	Chickenpox, Fungal Infection, Drug Reaction	Exact
Headache, Nausea, Visual disturbances	Migraine (Mayo Clinic)	Migraine, Vertigo, Malaria	Exact

The Table 6.1 shows the prediction verification results. Most cases showed exact or partial alignment with existing medical resources, indicating that the trained RF model generated predictions generally align with established medical knowledge. The prediction verification experiment assessed the correctness of the system's outputs by comparing them with expected diagnoses derived from trusted medical references (Centers for Disease Control and Prevention, 2023; Mayo Clinic, 2023; Cleveland Clinic, 2023). However, prediction accuracy decreased in cases involving ambiguous or overlapping symptoms, reflecting the inherent challenges of different diagnosis. This validation ensure that the system can effectively generate reasonable predictions, and it should be strictly used as a decision-making support tool and cannot replace professional medical assessments.

6.3 Web Application Development

The disease prediction system is implemented as a web-based application offers users with a convenient, user-friendly, and highly interactive interface. Its developed using a three-tier architecture design that include the frontend, backend, and database. This structure ensures the modularity, scalability and effective separation of responsibilities, facilitating maintenance and future extension.

6.3.1 Frontend Development

The system frontend is developed using React, with a component-based structure that is particularly well-suited for dynamic, interactive web application. The component-based structure ensures the consistency across the pages with reusable elements such as Header, Footer and so on. The core function of the frontend is to serve as the user interface, enabling seamless interaction with the system.

6.3.1.1 User Authentication and JWT Handling

Authentication is a critical component of the web application to ensure that only authorized users can access their history and predictions. The application utilizes JSON Web Tokens (JWT) for secure session management. Upon user login, the backend issues a JWT which is securely stored in local storage or session storage. When the frontend receives a user login request, it sends the login credentials to the backend API via a POST request. The backend validates the credentials and issues a JWT signed with a server-side secret key if successful.

If the user selects the “Remember Me” option, the token is stored in localStorage, ensuring persistence across browser sessions. Otherwise, the token is stored in sessionStorage, which expires automatically when the browser is closed. This ensures that sensitive endpoints such as fetching user details, submitting symptoms or viewing prediction histories are only accessible to authenticated users. The Figure 6.13 shows the code snippet of token storage logic that support both persistent and temporary sessions depending on user preferences.

```
if (formData.remember) {
  localStorage.setItem("token", data.token);
} else {
  sessionStorage.setItem("token", data.token);
}
```

Figure 6.13: Token Storage Logic

After the successful login, the frontend queries the backend for user profile details using the issued token. The returned information such as username is dynamically injected into the application state, allowing personalized display on the interface, including showing the username on the header. For each secured API request, the token is appended to the Authorization header. The Figure 6.14 shows the code snippet of token included in the Authorization header to authenticate protected API calls.

```
const resUser = await fetch("http://localhost:5000/api/user", {
  headers: { Authorization: `Bearer ${data.token}` }
});
```

Figure 6.14: Attach Token to Secured API Requests

On the backend, Flask validates the token and extracts the user’s identity. Only when the token is valid, the backend returns user specific data. This ensures that sensitive features remain accessible only to authenticated users.

6.3.1.2 Input Validation

The web application implements input validation to ensure the data integrity, prevent invalid input formats, and enhance security. Multiple forms in the disease prediction web application utilize client-side validation, verifying user input before submission to the backend. This design not only improves user experience through immediate feedback but also effectively reduces unnecessary server load.

6.3.1.2.1 Register Form Validation

The registration form includes multiple validation rules to ensure accurate data entry. These rules include checking for valid email formats, verifying password length requirements, and preventing users from submitting empty fields. The system provides immediate feedback to users through descriptive error messages, thereby enhancing user experience and reducing server load. When users register, the system will prompt users to enter the username, email address, date of birth, password, confirm password, and agree to the relevant terms and conditions. When the validation fails, the corresponding error message is displayed immediately on the form interface. The Figure 6.15 shows the code snippet of registration form validation. This logic prevents invalid input and enforces minimum password length requirements.

```
if (!formData.username.trim()) errors.username = "Username is required.";
if (!formData.email) {
  errors.email = "Email is required.";
} else if (!/\S+@\S+\.\S+/.test(formData.email)) {
  errors.email = "Enter a valid email address.";
}

if (!formData.dob) errors.dob = "Date of birth is required.";

if (!formData.password) {
  errors.password = "Password is required.";
} else if (formData.password.length < 6) {
  errors.password = "Password must be at least 6 characters.";
}

if (!formData.confirmPassword) {
  errors.confirmPassword = "Please confirm your password.";
} else if (formData.password !== formData.confirmPassword) {
  errors.confirmPassword = "Passwords do not match.";
}

if (!formData.agree) {
  errors.agree = "You must agree to the terms and conditions.";
}
```

Figure 6.15: Code Snippet for Registration Form Validation

6.3.1.2.2 Login Form Validation

The login form integrates multiple validation mechanisms to ensure secure and accurate user input. The primary checks include email validation and password validation. The system validated that the email field is not empty and conforms to a standard email pattern using a regular expression. If the input does not meet the requirements, an inline error message is displayed. For the password validation, the password field is required and cannot be blank. The missing input triggers an immediate validation error. If the user login successful, the system will redirect users to the home page. The Figure 6.15 shows the code snippet of login form validation. The below logic ensures that only correctly formatted data is sent to the backend, reducing the risk of invalid requests.

```
if (!formData.email) {
  newErrors.email = 'Email is required.'; formIsValid = false;
} else if (!validateEmail(formData.email)) {
  newErrors.email = 'Please enter a valid email address.'; formIsValid = false;
}
if (!formData.password) {
  newErrors.password = 'Password is required.'; formIsValid = false;
}
```

Figure 6.16: Code Snippet for Login Form Validation

6.3.1.2.3 Dropdown Symptom Selection Validation

The dropdown symptom input page integrates multiple validation checks to ensure that valid data is submitted to the machine learning model. Key validation features include search and filter, checkbox validation, and "Add" button validation. User can dynamically search symptoms by entering the keywords of the symptoms, with the system filtering the displayed list in real time. Each symptom can be selected via checkbox. The system prevents duplicated selections by filtering already added symptoms. When the user clicking the "Add" button, the system verifies that at least one symptom is checked. If no symptoms are selected, the alert appears on the interface. The Figure 6.17 shows the code snippet of the validation code for an empty symptom selection and an empty prediction request.

```
if (checked.length === 0) {
  alert("⚠ Please check at least one symptom before adding.");
  return;
}
if (selected.length === 0) {
  alert("⚠ Please add at least one symptom before predicting.");
  return;
}
```

Figure 6.17: Code Snippet for Dropdown Symptom Selection Validation

6.3.1.2.4 Free Text Symptoms Input Validation

The free text symptoms input page enables users to describe their health condition in natural language rather than selecting symptoms manually from a predefined list. Input validation ensures that the text is not empty before submitting. If the user attempts to extract symptoms without providing text, the system displays a warning message. After submission, the input text is sent to the backend via secure API request with JWT authentication. The backend applies large language model (Google Gemini) to identify and extract relevant symptom. The extracted symptoms are displayed in a dedicated section titled "Matched Symptoms" and users may remove the symptoms via a close icon. If the matched symptoms are empty, the system shows an alert to remind user to add at least one symptom. Figure 6.18 shows the code snippet for free text symptoms input validation.

```
if (!inputText.trim()) {
  alert("Please enter some text to describe your symptoms.");
  return;
}
```

```
if (matchedSymptoms.length === 0) {
  alert('Please add at least one symptom.');
  return;
}
```

Figure 6.18: Code Snippet for Free Text Symptoms Input Validation

6.3.1.2.5 Update Profile Form Validation

The update profile page enables authenticated users to modify their personal information, ensuring that their account details remain accurate and current. The system compares current input values with the original values retrieved from the backend. If no changes are detected, the system prevents unnecessary API calls by displaying a "No changes detected" alert. The username cannot be left blank, prevents accidental submission of an empty username. For the date of birth, the input restricted to valid date format. The gender section implemented a predefined options such as Male and Female, reducing the risk of inconsistent entries. The Figure 6.19 shows the code snippet for update profile form validation.

```
if (username !== originalUsername) updatedData.username = username;
if (dob !== originalDob) updatedData.dob = dob;
if (gender !== originalGender) updatedData.gender = gender;

if (Object.keys(updatedData).length === 0) {
  alert("No changes detected.");
  return;
}
```

Figure 6.19: Code Snippet for Update Profile Form Validation

6.3.1.2.6 Change Password Form Validation

The change password page enhances user account security by allowing users to update their password. This feature protects users' privacy and safeguards sensitive information stored in the system. The current password field, new password field, and confirm password field is mandatory. If left blank, the error message is displayed to prevent empty submissions. A minimum length requirement of 6 characters is required to encourage stronger password and reduce vulnerability. The system ensures that both new password and confirm password entries are match. Otherwise, the error message "Password do not match" is shown. The Figure 6.20 shows a code snippet for change password input validation.

```
if (!currentPassword.trim()) {
  newErrors.currentPassword = 'Current password is required.';
}
if (!newPassword.trim()) {
  newErrors.newPassword = 'New password is required.';
} else if (newPassword.length < 6) {
  newErrors.newPassword = 'New password must be at least 6 characters.';
}
if (!confirmPassword.trim()) {
  newErrors.confirmPassword = 'Please confirm your new password.';
} else if (newPassword !== confirmPassword) {
  newErrors.confirmPassword = 'Passwords do not match.';
}
```

Figure 6.20: Code Snippet for Change Password Input Validation

6.3.1.3 User Experience (UX) enhancements

6.3.1.3.1 Login and Register

For the inline error message, the validation errors are displayed directly below the corresponding input fields, guiding users to correct the errors. In addition, users can toggle between hiding and showing their password, improving usability while ensuring security. Users also can choose to persist their login session using either localStorage or sessionStorage in login page.

6.3.1.3.2 Dropdown Symptom Selection

The dropdown symptom input page was designed not only to ensure valid date entry but also to improve usability and provide an intuitive interaction flow for end users. A real-time search bar allows users to quickly filter the required symptoms from the massive symptom list by entering relevant keywords. This reduces cognitive load and ensures that users can efficiently locate the symptoms they intend to select.

Furthermore, users can check multiple symptoms before confirming their selection. Once the symptoms are added, the selected symptoms are clearly displayed in a dedicated “Selected Symptoms” section. The selected symptom can be removed using a close icon, giving users full control to revise their selection without having to restart the process. The system also ensures the clear navigation control. The button such as “Back”, “Add” and “Predict” are clearly labelled and visually distinct, reducing ambiguity in navigation.

6.3.1.3.3 Free Text Symptom Input

The free text symptom input page provides guidance through placeholder text. The input field contains a placeholder sentence such as “I have a fever and cough for 2 days…” to guide users on how to describe their symptoms. This page also includes loading feedback. During symptom extraction, the “Extract Symptoms” button will be temporarily disabled and labelled “Extracting…” to prevent duplicate requests and indicate to the user that the system is processing. By allowing users to remove symptoms from the extracted symptom list, the system provides flexibility and avoid forcing incorrect inputs. If no symptoms are selected, the user is informed through an alert. The navigation buttons like “Back” and “Predict” are styled consistently with other input methods.

6.3.1.3.4 Update Profile

The user information such as username, date of birth and gender is automatically retrieved and displayed when the page loads. This can save users time and effort. Only modified fields are sent to backend, minimizing server load and preventing unnecessary overwriting of unchanged data. A confirmation message “Profile updated successfully” is displayed upon successful update. If the username is changed, the update is immediately reflected across the web application through the global state update. Users can return to the profile page at any time using the “Back” button, ensure the smooth navigation.

6.3.1.3.5 Change Password

The contextual message like “Hi, [username]” fosters user engagement. The form validation runs upon submission, immediately highlighting missing or invalid fields to avoid wasting server requests. When the update request is in progress, the “Update Password” button changes to “Updating…” and disabled to prevent duplicate submissions. A Back button enables users to easily return to the profile page. For successful update, the system displays a confirmation alert with the backend response message and redirect users to profile page. If the password is incorrect, a clear error message is displayed. The error messages and button states improve usability by guiding users step by steps.

6.3.1.3.6 Result

The result page displayed the match strength indicators via icons. The predictions are visually ranked using text and star ratings. For the strongly matched disease show full star, for the moderately matched disease show half of star, and the weakly matched disease show colourless star. This gives users confidence levels in the prediction without needing to understand probabilities machine learning outputs. The results page also displayed clear card-based layout. Two distinct cards separate “Potential Disease” and “Your Symptoms”, This layout helps users easily connect to their input and prediction. The “View Medical Advice” button provides a clear next step, guiding uses to additional information. The system displays “Loading results…” while predictions are being processed, keeping the UI responsive. It also includes health disclaimer for trust and responsibility. This enhances trustworthiness and ensures ethical communication of AI results.

6.3.1.3.7 General Medical Advice

The general medical advice page is designed with a strong emphasis on user experience, ensuring the information is presented in a structured, clear, and easily accessible manner. The system avoids displaying vague generalizations or raw data, instead organizing recommendations into distinct sections such as disease descriptions, lifestyle advice, prevention strategies, and guidance on when to seek medical attention. Users can simultaneously view the correlation between entered symptoms and the system's predicted conditions. This effectively builds user trust in the recommendations provided by the system. Additionally, users can save predictions and recommendations to their personal history for future reference, ensuring continuity in healthcare services. The page also includes a clear disclaimer emphasizing that the advice provided is for informational purposes only and should not replace professional medical staff.

6.3.1.3.8 History

The History page provides users with a clear and organized view of past predictions, covering symptoms, conditions, and medical advice. Featuring a card-based layout for enhanced readability, it uses icons for quick recognition and allows users to expand or collapse detailed recommendations as needed. Functions such as confirmed record deletion, real-time feedback notifications, and token-based secure access further optimize usability and control. This design not only elevates the overall user experience but also ensures users can conveniently and securely review and manage their health prediction records.

6.4 Backend Development

The backend of the Disease Prediction Web Application using Machine Learning was developed using Flask, a lightweight Python web framework well-suited for building RESTful APIs. The backend serves as the communication layer between frontend and machine learning model, handling user authentication, symptom extraction, disease prediction, general medical advice generation and database operations. The backend also integrates with MySQL database. The backend defines all the API endpoints, loads the trained Random Forest model, the random_forest_model.pkl, and manages middleware such as JWT authentication and CORS configuration. Sensitive environment variables such as the secret key, Gemini API key and database setup are managed using python-dotenv, ensuring secure configuration management. Figure 6.21 shows the code snippet of project setup.

```
load_dotenv()

app = Flask(__name__)
CORS(app, origins=["http://localhost:3000"])
SECRET_KEY = os.getenv("SECRET_KEY")
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
MODEL_DIR = os.path.join(BASE_DIR, "ML_model")
```

Figure 6.21: Code Snippet of Project Setup

6.4.1 Project Architecture

The key aspects of the architecture include authentication and security, database integration, machine learning integration, prediction history and medical advice, and RESTful design.

6.4.1.1 JWT Decorator

The system implemented using JWT tokens with custom decorators to protect the sensitive routes such as profile, predictions and history. If no token is found, return a JSON response with a 401 Unauthorized status and an error message “Token is missing”. Figure 6.22 illustrates the code snippet for JWT Decorator.

```python
# --------------------- JWT DECORATOR --------------------
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        if "Authorization" in request.headers:
            auth_header = request.headers["Authorization"]
            if auth_header.startswith("Bearer "):
                token = auth_header.split(" ")[1]

        if not token:
            return jsonify({"error": "Token is missing!"}), 401

        try:
            data = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
            current_user_id = data["user_id"]
        except jwt.ExpiredSignatureError:
            return jsonify({"error": "Token has expired! Please log in again."}), 401
        except jwt.InvalidTokenError:
            return jsonify({"error": "Invalid token!"}), 401

        return f(current_user_id, *args, **kwargs)
    return decorated
```

Figure 6.22: Code Snippet for JWT Decorator

6.4.1.2 Database Integration

The MySQL database is used to persist user information, prediction history, symptoms, diseases, and general medical advice. Database operations are abstracted through a db_connection,py module for cleaner code management. The get_connection() function established a connection to MySQL database using environment variables for configuration. Figure 6.23 shows a code snippet of the get_connection() function.

```python
def get_connection():
    return mysql.connector.connect(
        host=os.getenv("DB_HOST"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"),
        database=os.getenv("DB_NAME")
    )
```

Figure 6.23: get_connection() function

6.4.1.3 Machine Learning Integration

The backend loads pretrained models and feature encodings via Joblib, enabling real-time predictions from user inputs. Prediction made using either structured symptom selection like dropdown, or unstructured free-text input processed with a symptom extraction pipeline powered by external NLP helpers like gemini_helper. The extract_symptoms() function in gemini_hepler.py is used to extract the user free-text input by using Google Gemini API and return the symptoms that exist in the provided list. Figure 6.24 shows the code snippet for loading machine learning model. Figure 6.25 shows the code snippet of the extract symptoms function.

```
# --------------------- LOAD ML MODEL ----------------------
MODEL_PATH = os.path.join(MODEL_DIR, "random_forest_model.pkl")
FEATURES_PATH = os.path.join(MODEL_DIR, "symptom_features.pkl")

model = joblib.load(MODEL_PATH)
```

Figure 6.24: Code Snippet for Load ML Model

```
# --- Gemini extraction (free-text) ---
def extract_symptoms(user_text):
    prompt = f"""
    User input: "{user_text}"

    Valid symptoms: {SYMPTOMS}

    Task:
    - Identify only the symptoms explicitly present or clearly misspelled in the input.
    - Correct typos if needed.
    - Do NOT add extra symptoms that are not supported by the input.
    - Only return symptoms that exist in the provided list.
    - Respond ONLY with a valid JSON array (e.g., ["cough", "high_fever"]).
    """

    response, model_used = safe_generate_content(prompt)
    raw_text = response.candidates[0].content.parts[0].text.strip()
    try:
        parsed = json.loads(raw_text)
        if isinstance(parsed, list):
            return [s for s in parsed if s in SYMPTOMS], model_used
        if isinstance(parsed, dict) and "symptoms" in parsed:
            return [s for s in parsed["symptoms"] if s in SYMPTOMS], model_used
        return [], model_used
    except Exception as e:
        print("Gemini parsing error:", e, "Raw:", raw_text)
        try:
            if "[" in raw_text and "]" in raw_text:
                array_str = raw_text[raw_text.find("["): raw_text.rfind("]")+1]
                symptoms = json.loads(array_str)
                if isinstance(symptoms, list):
                    return [s for s in symptoms if s in SYMPTOMS], model_used
        except Exception as inner_e:
            print("Fallback also failed:", inner_e)
        return [], model_used
```

Figure 6.25: Extract Symptoms Function

6.4.2 API Endpoints

The backend exposes multiple RESTful API endpoints, categorized into user management, prediction, history and general medical advice. Each endpoint features a lightweight design with security capabilities and follows consistent request-response structures. JWT authentication is required for protected endpoints to ensure authorized access.

6.4.2.1 User Authentication and Profile Management

- **POST /api/register**
  Registers a new user by storing their credentials and basic profile data in the database. Input validation ensures all required fields are provided such as username, email, password, and data of birth. Passwords are securely stored using Werkzeug's hashing mechanism.
- **POST /api/login**
  Authenticates a user by verifying the provided email and password. Upon successful validation, a JWT token is issued, which must be attached to subsequent requests for protected endpoints.
- **GET /api/user (Protected)**
  Returns the authenticated user's ID and username based on the JWT token. Used for session validation and personalization on the frontend.
- **GET /api/profile (Protected)**
  Retrieves detailed profile information such as username, email, date of birth, gender for the authenticated user.
- **PUT /api/profile (Protected)**
  Allows users to update selected profile fields such as username, date of birth, or gender. Partial updates are supported through dynamic query construction.
- **PUT /api/change-password (Protected)**
  Enables users to securely update their password after validating their existing password. New passwords are hashed before storage.

6.4.2.2 Symptom and Disease Management

- **GET /api/symptoms**
  Fetches the complete list of symptoms stored in the database, enabling the frontend dropdown selection method.
- **GET /api/diseases**
  Retrieves all disease records from the database. This is used to maintain consistency between predictions and stored disease references.
- **POST /api/extract-symptoms (Protected)**

Processes free-text input and extracts symptom entities using the Gemini Helper NLP module. This supports the unstructured input method and ensures that symptom names align with the system's knowledge base.

- **POST /api/predict (Protected)**
  Predicts potential diseases based on user input. Two input methods are supported by the system:
  - **Dropdown-based**: The user selects symptoms from a predefined list.
  - **Free text**: The user provides natural language descriptions, which are processed into structured symptoms before prediction.

  Predictions are generated using the pre-trained Random Forest model and returned with ranked match strengths.

6.4.2.3 Medical Advice Generation

- **POST /api/advice (Protected)**
  Accepts a list of diseases and returns general medical advice for each. Advice is generated dynamically using the Gemini Helper module. This provides contextual recommendations to users while reinforcing the disclaimer that the advice is not a substitute for professional medical consultation.

6.4.2.4 Prediction History

- **POST /api/history (Protected)**
  Saves a prediction history to the database, including selected symptoms, predicted diseases, and corresponding general medical advice. Missing symptoms or diseases not found in the database are also recorded for consistency checks.
- **GET /api/history (Protected)**
  Retrieves all historical prediction records for the authenticated user, including symptoms, predicted diseases, associated advice, and timestamps. Results are ordered by prediction date (latest first).
- **DELETE /api/history/<id> (Protected)**

Deletes a specific prediction history record belonging to the authenticated user. The related records such as predicted symptoms and diseases are also removed to maintain referential integrity.

6.4.2.5 API Endpoint Overview

The Table 6.1 shows the API Endpoint Overview.

Table 6.2: API Endpoints Overview

Endpoint	Method	Description	Auth Required
/api/register	POST	Registers a new user with username, email, password, and DOB. Stores hashed password securely.	No
/api/login	POST	Authenticates user with email and password, returns JWT token on success.	No
/api/user	GET	Retrieves authenticated user's ID and username for session validation.	Yes
/api/profile	GET	Fetches detailed profile (username, email, DOB, gender).	Yes
/api/profile	PUT	Updates profile details (username, DOB, gender). Partial updates supported.	Yes
/api/change-password	PUT	Allows user to change password after verifying old password.	Yes

/api/symptoms	GET	Returns all symptoms stored in the system database.	No
/api/diseases	GET	Returns all diseases stored in the system database.	No
/api/extract-symptoms	POST	Extracts symptoms from free-text input using Gemini Helper NLP.	Yes
/api/predict	POST	Predicts potential diseases based on symptoms. Uses ML model for ranked results.	Yes
/api/advice	POST	Provides general medical advice for one or more diseases using Gemini Helper.	Yes
/api/history	POST	Saves prediction session (symptoms, diseases, advice) into history.	Yes
/api/history	GET	Retrieves all past prediction history for the authenticated user.	Yes
/api/history/<id>	DELETE	Deletes a specific prediction history record.	Yes

6.4.3 Implementation of Google Gemini API

The backend also integrates Google Gemini API, enabling context-aware medical advice based on disease prediction and symptom extraction. The Gemini API was primarily utilized to extract symptoms from free-text input, generate general medical advice, and provide personalize output. It provides clear and user-friendly guidance on possible treatment or prevention tips after disease prediction. Prediction model (Random Forest model) identifies possible diseases based on symptoms, while Gemini generates structured, user-friendly guidance presented as prevention and lifestyle recommendations. This ensures that the application not only delivers predictive results but also provides actionable next steps, thereby creating greater value for end users.

The objective of integrating the Google Gemini API into the backend is to enhance user value, improve consistency and reliability, ensure user safety, and achieve frontend compatibility. The Google Gemini API not only generates predictive results but also delivers structured health advice to guide users improving their lifestyle and preventing diseases. In addition, enforces a structured JSON output format to ensure consistency across all diseases. It also provides only general and non-diagnostic information, focusing on health awareness, disease prevention, and guidance on when to seek professional medical assistance. The Gemini API returns data in a machine-readable JSON format that can be directly called and rendered by the frontend application.

6.4.3.1 Prompt Engineering and Structured Output

The function get_general_advice_for_multiple(diseases) was developed to interact with the Google Gemini. This function constructs a carefully engineered prompt that instructs Google Gemini return advice in a strict JSON format. For each predicted disease, Gemini is requested to provide four specific fields, including description, lifestyle tips, prevention tips and a guideline on when to seek care. This structure not only improves readability but also ensures that the generated content can be validated, parsed, and integrated into the web application workflow.

As discussed in Chapter 5 (Prompt Design Study), the Role-based prompting strategy demonstrated the most effective performance. It consistently produced well-balanced and guideline-consistent outputs with accuracy, clarity, and completeness as well as maintaining reasonable response time. Therefore, role-based prompting was adopted in the Google Gemini API implementation to generate structured medical advice for predicted diseases. By explicitly assigning the model the role of a health assistant, the generated responses were not only aligned with medical communication standards but also returned in a consistent JSON format. This structured representation enables seamless integration with the backend pipeline, ensuring that the generated advice could be directly parsed, validated, and displayed within the web application without requiring extensive post-processing. Figure 6.26 shows a code snippet for generate structure medical advice via Google Gemini API.

```python
# --- Advice from Gemini (Structured JSON with prevention) ---
def get_general_advice_for_multiple(diseases):
    if not diseases:
        return {}, None

    prompt = f"""
    You are a health assistant.

    For each of the following conditions:
    {', '.join(diseases)}

    Return a JSON object where:
    - Key = disease name
    - Value = another object with 4 fields:
      "description": short description of the condition (2-3 sentences, max 50 words)
      "lifestyle_tips": array of 2-3 simple lifestyle tips (short phrases)
      "prevention_tips": array of 2-3 prevention tips (short phrases)
      "when_to_seek_care": one clear line telling when to see a doctor

    Example format:
    {{
      "Malaria": {{
        "description": "...",
        "lifestyle_tips": ["...", "..."],
        "prevention_tips": ["...", "..."],
        "when_to_seek_care": "...."
      }}
    }}

    Do not include any text outside the JSON.
    """

    response, model_used = safe_generate_content(prompt)
    raw_text = response.candidates[0].content.parts[0].text.strip()
```

Figure 6.26: Generate Structured Medical Advice via Google Gemini API

CHAPTER 7

SYSTEM TESTING

7.1 Introduction

System testing is a critical phase in the software development lifecycle that is utilized to ensure the disease prediction web application functions as intended and meets both functional and non-functional requirements. This chapter outlines the details of testing strategies, methodologies, and results for validating system performance, usability, and reliability.

The system integrates machine learning-based disease prediction functionality, a symptom extraction and medical advice module powered by Google Gemini, and an architecture utilizing a Flask backend with a fully interactive React frontend architecture. Testing ensures all modules are compatible with each other, the system security remains uncompromised, and users can access all intended features without errors.

7.2 Unit Testing

Unit testing focuses on verifying the correctness of individual components or functions within a system.

7.2.1 Registration Feature

Table 7.1: Unit Test Case for Registration Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC01	Validate empty username	Leave username blank and submit form	username: “”, email: “test@gmail.com”, password: “123456”, confirmPassword: “123456”, dob: “2000-01-01”, agree: true	Error message “Username is required.” displayed	Pass
UTC02	Validate invalid email format	Enter invalid email and submit	username: “John”, email: “johngmail”, password: “123456”, confirmPassword: “123456”, dob: “2000-01-01”, agree: true	Error message “Please enter a valid email address.” displayed	Pass
UTC03	Validate empty email	Leave email empty and submit	username: “John”, email: “”, password: “123456”,	Error message “Email is required.” displayed	Pass

			confirmPassword: “123456”, dob: “2000-01-01”, agree: true		
UTC04	Validate empty DOB	Leave DOB blank and submit	username: “John”, email: “john@gmail.com”, password: “123456”, confirmPassword: “123456”, dob: “”, agree: true	Error message “Date of birth is required.” displayed	Pass
UTC05	Validate short password	Enter password <6 characters	username: “John”, email: “john@gmail.com”, password: “123”, confirmPassword: “123”, dob: “2000-01-01”, agree: true	Error message “Password must be at least 6 characters.” displayed	Pass
UTC06	Validate mismatched password	Enter different password and confirm password	username: “John”, email: “john@mail.com”, password: “123456”, confirmPassword: “654321”, dob: “2000-01-01”, agree: true	Error message “Passwords do not match.” displayed	Pass

UTC07	Validate empty password	Leave password empty and submit	username: “John”, email: “john@gmail.com”, password: “”, confirmPassword: “654321”, dob: “2000-01-01”, agree: true	Error message “Password is required.” displayed	Pass
UTC08	Validate terms agreement	Terms agreement checkbox not checked	username: “John”, email: “john@gmail.com”, password: “123456”, confirmPassword: “123456”, dob: “2000-01-01”, agree: false	Error message “You must agree to the terms and conditions.” displayed	Pass
UTC09	Show/Hide password toggle	Click show/hide toggle for password and confirm password fields	-	Password visibility toggles correctly between plain text and hidden	Pass
UTC10	Successful frontend validation	Enter valid inputs	username: “John”, email: “john@gmail.com”, password: “123456”, confirmPassword: “123456”, dob: “2000-01-01”, agree: true	Form submits successfully, calls backend API	Pass

UTC11	Duplicate email error	Enter existing email	username: “John”, email: “john@gmail.com”, password: “123456”, confirmPassword: “123456”, dob: “2000-01-01”, agree: true	Error message: “Email already exists” displayed	Pass
UTC12	Redirect to Login page	Click “Already have an account? Log in”	-	Navigated to /login page	Pass

7.2.2 Login Feature

Table 7.2: Unit Test Case for Login Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC13	Validate empty email	Leave email blank and click Login	email: “”, password: “123456”,	Error message “Email is required.” displayed	Pass
UTC14	Validate invalid email format	Enter email without proper format	email: “johngmail”, password: "123456",	Error message “Please enter a valid email address.” displayed	Pass

UTC15	Validate empty password	Enter valid email but leave password	email: "john@gmail.com", password: "",	Error message "Password is required." displayed	Pass
UTC16	Validate login with incorrect credentials	Enter wrong email/password and submit	email: "john@gmail.com", password: "wrongpass",	Alert popup: "Invalid credentials"	Pass
UTC17	Validate login with correct credentials (Remember unchecked)	Enter valid email & password, leave Remember Me unchecked	email: "john@gmail.com", password: "123456"	JWT token stored in sessionStorage, redirected to /home	Pass
UTC18	Validate login with correct credentials (Remember checked)	Enter valid email & password, check Remember Me	email: "john@gmail.com", password: "123456"	JWT token stored in localStorage, redirected to /home	Pass
UTC19	Validate user data fetch after login	Successful login, then fetch /api/user with Bearer token	email: "john@gmail.com", password: "123456"	setUsername() and setUserId() updated with correct values	Pass

UTC20	Toggle password visibility (Show to Hide)	Click “Show” button in password field	password: “123456”	Password input changes type from password to text and button label changes to Hide	Pass
UTC21	Toggle password visibility (Hide to Show)	Click “Hide” button in password field	password: “123456”	Password input changes type from text to password and button label changes to Show	Pass
UTC22	Redirect to Register page	Click “Don’t have an account? Sign Up”	-	Navigated to /register page	Pass

7.2.3 Profile

Table 7.3: Unit Test Case for Profile

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC23	Validate profile fetch with valid token	Store valid token in localStorage, open Profile page	token=valid	Profile data fetched and displayed (username, email, dob, gender)	Pass

UTC24	Validate profile fetch with missing token	Remove token from storage, open Profile page	token=none	Failed to fetch profile	Pass
UTC25	Validate change password navigation	Click "Want to change password? Click me!" link	-	Redirected to /change-password page	Pass
UTC26	Validate "Update Profile" navigation	Click "Update Profile" button	-	Redirected to /update-profile page	Pass
UTC27	Validate "Back" navigation	Click "Back" button	-	Redirected to /home page	Pass
UTC28	Validate Date of Birth formatting	Profile response contains DOB in ISO format	contains DOB in ISO format dob="1999-10-05"	DOB displayed as 05 Oct 1999	Pass

7.2.4 Update Profile Feature

Table 7.4: Unit Test Case for Update Profile Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC29	Verify profile data loads on page load	Open Update Profile page after view profile	User existing profile	Profile fields (username, DOB, gender) are pre-filled correctly	Pass
UTC30	Update username only	Change username and click Update	New username: John123	Alert “Profile updated successfully!” Redirect to Profile page	Pass
UTC31	Update date of birth only	Change DOB and click Update	New DOB: 2005-05-05	Alert “Profile updated successfully!” Redirect to Profile page	Pass
UTC32	Update gender only	Change gender from Male to Female and click Update	Gender: Female	Alert “Profile updated successfully!” Redirect to Profile page	Pass

UTC33	Cancel update and click Back	Click Back button instead of Update	-	User redirected back to Profile page	Pass
UTC34	Check invalid username entry (empty string)	Clear username field, Click Update	username: “”	System should still allow DOB/Gender changes, but username remains unchanged	Pass

7.2.5 Change Password Feature

Table 7.5: Unit Test Case for Change Password Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC35	Verify all fields required	Leave all inputs empty and click “Update Password”	-	Errors: “Current password is required.”, “New password is required.”, “Please confirm your new password.”	Pass

UTC36	Validate new password length	Enter valid current password but short new password (<6 chars)	Current: john123, New: 123, Confirm: 123	Error: “New password must be at least 6 characters.”	Pass
UTC37	Validate password mismatch	Enter mismatched new and confirm password	Current: john123, New: 123456, Confirm: 654321	Error: “Passwords do not match.”	Pass
UTC38	Successful password change	Enter valid data and click Update	Current: john123, New: 123456, Confirm: 123456	Alert success message, Redirect to /profile	Pass
UTC39	Invalid current password	Enter wrong current password	Current: john12345, New: 123456, Confirm: 123456	Alert: “Old password is incorrect”	Pass
UTC40	Back button navigation	Click Back button instead of submitting	-	Redirects user back to /profile without changes	Pass

7.2.6 Select Input Method Feature

Table 7.6: Unit Test Case for Select Input Method Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC41	Validate page loads correctly	Open Select Input page	-	Page shows title, instruction, and two options (Dropdown and Free Text)	Pass
UTC42	Validate dropdown option navigation	Click anywhere on Dropdown option card or button	-	User navigates to /dropdown	Pass
UTC43	Validate free text option navigation	Click anywhere on Free Text option card or button	-	User navigates to /free-text	Pass
UTC44	Responsive layout	Resize window to small screen	-	Cards remain responsive and readable	Pass
UTC45	Back button navigation	Click Back button instead of submitting	-	Redirects user back to /profile without changes	Pass

7.2.7 Dropdown Input Feature

Table 7.7: Unit Test Case for Dropdown Input Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC46	Validate search filters symptom list	Type “fever” in search bar	Search = fever	Only symptoms matching “fever” appear	Pass
UTC47	Validate symptom checkbox selection	Check a symptom (e.g., “Fever”)	Symptom: Fever	Checkbox marked, symptom stored in checked	Pass
UTC48	Add selected symptoms	Select multiple symptoms, Click Add	Selected: Fever, Vomiting	Symptoms appear under “Selected Symptoms”	Pass
UTC49	Prevent adding empty selection	Click Add without selecting	-	Alert: “Please check at least one symptom before adding.”	Pass
UTC50	Remove symptom from selected	Click “X” on a selected symptom	X Fever	Symptom removed from selected list	Pass
UTC51	Successful prediction	Select symptoms to Click Predict	Fever, Cough	Navigate to /result with predictions passed in state	Pass

UTC52	Prevent duplicate selected symptoms	Select same symptom twice	Fever, Fever	Symptom only appears once in selected list	Pass

7.2.8 Free Text Feature

Table 7.8: Unit Test Case for Free Text Input Feature

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC53	Prevent empty submission	Click “Extract Symptoms” with empty textarea	“”	Alert: “Please enter some text to describe your symptoms.”	Pass
UTC54	Successful symptom extraction	Enter valid input and extract	“I have fever and cough”	Extracted symptoms appear in “Matched Symptoms” list	Pass
UTC55	No symptoms extracted	Enter unrelated text to Extract	“I like pizza”	Alert: “No symptoms extracted. Please try again.”	Pass
UTC56	Remove matched symptom	Click “X” on a matched symptom	X Fever	Symptom removed from list	Pass

UTC57	Successful prediction	Enter text, extract, then predict	“I have fever and cough”	Navigate to /result with predictions passed in state	Pass
UTC58	Loading state behavior	Click Extract Symptoms	“I have fever	Button text changes to “Extracting…” until done	Pass

7.2.9 Prediction Result

Table 7.9: Unit Test for Prediction Result

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC59	Validate results load correctly	Navigate to Result page with results in state	Backend: results=[{disease:"Flu"}]	Diseases displayed in card	Pass
UTC60	Validate symptoms display	Navigate to Result page with symptoms in state	symptoms= ["Fever","Cough"]	Symptoms listed in card	Pass
UTC61	Validate match strength icons	Load results with multiple diseases	3 diseases	Star icons display correctly (Strong, Moderate, Weak)	Pass
UTC62	Navigate to medical advice	Click “View Medical Advice”	results + symptoms available	Navigate to /medical-advice with data passed	Pass

7.2.10 General Medical Advice

Table 7.10: Unit Test Case for General Medical Advice

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC63	Validate symptoms list	Navigate with symptoms in state	["Fever","Cough"]	Symptoms listed	Pass
UTC64	Fetch advice successfully	Navigate with results	Flu → Backend returns advice	Advice cards displayed	Pass
UTC65	Handle no advice	Backend returns empty advice	Flu → No advice	"No advice available" shown	Pass
UTC66	Save prediction successfully	Click "Save Prediction" with valid results	results + symptoms	Successful Message: "Prediction and advice saved successfully!" Redirect to History	Pass

7.2.11 Historical

Table 7.11: Unit Test Case for Historical

Unit Test Case ID	Test Case Description	Test Procedure	Test Data	Expected Result	Status
UTC67	Fetch history successfully	Navigate to page with valid token	Records available	Cards display with date, symptoms, diseases and advice	Pass
UTC68	Handle empty history	Navigate with no records	None	"No historical records found." shown	Pass
UTC69	Redirect if no token	Clear token, then navigate	-	Redirected to login with error message	Pass
UTC70	Delete history record	Click delete icon on a record	Record ID	Record removed + success message	Pass
UTC71	Delete cancelled	Click delete, cancel confirm	-	Record not deleted	Pass

UTC72	Expand advice text	Click “Read More” on long advice	Advice text > 150 chars	Full text shown	Pass
UTC73	Collapse advice text	Click “Show Less” after expansion	Advice text > 150 chars	Text collapses	Pass
UTC74	Home button navigation	Click “Home”	-	Navigate to home page	Pass

7.3 Integration Testing

Integration testing ensures that different modules of the disease prediction web application can work together seamlessly to form a unified system. Unlike unit testing, which verifies individual components, integration testing focuses on interactions between frontend interfaces and backend services. This phase validates whether data flows correctly between pages, tokens are securely handled, and results are consistently stored and retrieved. By implementing integration testing, potential issues such as API mismatches, data processing errors, or session management failures can be identified and resolved before system deployment.

Table 7.12: Integration Test Cases

Integration Test Case ID	Test Case Description	Test Procedure	Expected Result	Status
ITC01	Verify registration integrates with backend authentication.	1. Navigate to registration page. 2. Enter valid details. 3. Submit. 4. Login using new account.	User is successfully registered and able to log in.	Pass
ITC02	Verify login with token allows access to protected pages (e.g Profile).	1. Login with valid credentials. 2. Navigate to Profile. 3. Refresh page.	Profile details are displayed using stored token; session persists.	Pass
ITC03	Verify password change updates backend and login validation.	1. Login. 2. Change password. 3. Logout. 4. Login with old password (fail). 5. Login with new password (success).	Old password rejected; new password accepted.	Pass
ITC04	Verify dropdown symptom selection integrates with disease prediction.	1. Navigate to symptom input page. 2. Select symptoms. 3. Submit.	Backend returns prediction; results displayed correctly.	Pass

ITC05	Verify integration between Result page and Medical Advice page.	1. From Result, click “View Medical Advice”.	Medical advice is retrieved for predicted disease and displayed.	Pass
ITC06	Verify saving prediction stores record in History.	1. On Medical Advice page, click “Save Prediction”. 2. Navigate to History.	Prediction is stored and visible in History with correct details.	Pass
ITC07	Verify History page fetches consistent saved data.	1. Login. 2. Navigate to History.	All previously saved predictions are displayed correctly.	Pass
ITC08	Verify deletion of record updates backend and frontend.	1. Navigate to History. 2. Delete a record. 3. Refresh page.	Deleted record no longer appears.	Pass
ITC09	Verify session expiration handling when token is invalid.	1. Login. 2. Clear/expire token. 3. Navigate to Profile/History.	User redirected to Login with session expired message.	Pass
ITC10	Verify end-to-end workflow from registration to history.	1. Register new user. 2. Login. 3. Input symptoms. 4. Predict disease.	All steps succeed; saved prediction is available in History.	Pass

		5. View advice. 6. Save prediction. 7. Navigate to History.		

7.4 User Acceptance Testing (UAT)

User Acceptance Testing (UAT) is actual end users evaluated the system to ensure that the system meets the user needs, requirements, and expectations. The core objective of UAT is to validate whether the disease prediction web application using machine learning functions correctly in real world scenarios and delivers expected results. Unlike Unit Testing and Integration Testing, UAT emphasizes usability, output correctly, and alignment with user expectations.

This project conducted a UAT by distributing structured Google Forms questionnaires to a group of 30 users to perform hands-on testing of the system. They evaluated core functional modules, including register and login operation, symptom input via both dropdown menus and free text fields, disease prediction, viewing medical recommendations, and accessing historical records. Participants were asked to independently test the web application and provide feedback based on their user experience. The full Google Form questionnaire is attached in Appendix A for reference.

7.4.1 User Acceptance Testing Result

Table 7.13 shows a User Acceptance Testing Result summary.

Table 7.13: User Acceptance Testing Result.

Question	Aspect Evaluated	3(n)	4(n)	5(n)	Total	Average Rating
Q1	Ease of navigation	3	11	16	30	4.43
Q2	Ease of inputting symptoms	4	14	12	30	4.26
Q3	Clarity of error messages	3	4	18	30	4.53
Q4	System generated predictions	3	10	17	30	4.46
Q5	Accuracy of predictions	5	12	13	30	4.26
Q6	Relevance of medical advice	6	9	15	30	4.30

Q7	Accessibility of history feature	3	11	16	30	4.43
Q8	Overall satisfaction	3	11	16	30	4.43
Q9	No major difficulties encountered	1	12	17	30	4.53

The results indicate that the system achieved the anticipated usability goals, with 90% of users rating most aspects at 4 points or higher. The highest-rated aspects were clarity of error messages (4.53) and no major difficulties encountered (4.53), indicating that the system is straightforward to operate and easy to interact with. The only slightly lower score was for prediction accuracy, suggesting that while the predictions are generally acceptable, users may expect higher precision.

The moderately rated aspects included ease of inputting symptoms (4.26), accuracy of predictions (4.26), and relevance of medical advice (4.30). While these scores are still positive, they suggest that users see room for improvement in these domains. In particular, symptom input enables smoother operation, while leveraging larger datasets and more advanced models further enhances prediction accuracy and recommendation relevance.

Open-ended feedback (Q10) provided additional suggestions such as adding more advice information, adding dashboard page, further streamlining the layout, add multilingual support, include more healthcare information and so on. The full results can be found in Appendix B.

Based on the UAT results, the Disease Prediction Web Application using Machine Learning is considered user-friendly, functionally robust, and aligned with the user requirements. While prediction accuracy can be enhanced and medical advice content expanded through fine tuning, the system has successfully achieved the objectives of the user acceptance testing.

7.5 User Interface Design Feedback

To evaluate the usability and overall design of the system, a User Interface Design Feedback survey via Google Forms was conducted. This survey aimed to assess the system interface's intuitiveness, clarity and visual appeal while identifying areas for improvement. Respondents were asked to provide the feedback based on their actual usage experience. Most questions were utilized a 5-point Likert scale, where 1 represented "Strongly Disagree" and 5 represented "Strongly Agree." The full Google Form questionnaire is attached in Appendix C for reference. Table 7.13 shows a User Interface Design Feedback summary.

Table 7.14: User Interface Design Feedback

Question	**Aspect Evaluated**	**3(n)**	**4(n)**	**5(n)**	**Total**	**Average Rating**
Q1	Ease of use without training	2	10	18	30	4.53
Q2	Intuitiveness of UI design	1	15	14	30	4.43
Q3	Visual appeal of design	4	8	18	30	4.30
Q4	Colour comfort and theme consistency	5	8	17	30	4.40
Q5	Clarity of labels and buttons	3	10	17	30	4.46
Q6	Navigation and menu usability	1	13	16	30	4.5
Q7	Responsiveness and performance	6	8	16	30	4.33
Q8	Clarity of system feedback/error messages	3	13	14	30	4.36
Q9	User confidence in performing tasks	1	9	20	30	4.63

User Interface evaluation results indicate that users were generally satisfied, with all average ratings above 4.3 out of 5, reflecting a strong level of acceptance. User confidence in task execution (4.63) received the highest rating, indicating that users felt comfortable and capable when interacting with the system. Similarly, ease of use without training (4.53) and navigation and menu usability (4.50) also scored exceptionally high, demonstrating that the design is intuitive and requires minimal learning effort. Other aspects such as clarity of labels and buttons (4.46) and intuitiveness of UI design (4.43) also received high ratings, highlighting effective design choices that support smooth interaction. Meanwhile, the ratings for visual appeal of design (4.30) and responsiveness and performance (4.33) were slightly lower than other categories, though still positive.

Open-ended feedback (Q10) provided additional suggestions such as add dark mode for better accessibility, provide dashboard page, add tooltips or hints for new users, use more visuals for results, improve spacing and alignment of elemetns and so on. The full results can be found in Appendix D. All these results demonstrate that the web application provides a smooth and intuitive user experience. Respondents also expressed high confidence in using the interface for disease prediction tasks.

Overall, the survey indicated that the system's user interface design is intuitive and easy to use, aligning with its intended purpose and achieving usability objectives of supporting users in efficiently and confidently utilizing the disease prediction functionality.

CHAPTER 8

CONCLUSION AND RECOMMENDATION

8.1 Introduction

This conclusion outlined the achievement of the project objectives, limitation of the project and valuable suggestions into possible future work. The Disease Prediction Web Application using Machine Learning has been successfully developed and tested. This system provides a user-friendly, secure and efficient platform for predicting diseases based on user-provided health data.

In the initial of the project, it is essential to clearly define the problem statements, project objective, project scope and reviewing the existing similar applications to collect functional requirements and non-functional requirements. In the development phase, the system integrates both dropdown symptom selection and free text input method to accommodate diverse user needs. The backend was implemented using Flask, while the frontend was built with React, ensuring both scalability and responsiveness.

Following development and implementation, extensive testing was conducted, including unit testing, integration testing, user acceptance testing (UAT), and user interface design feedback collection. The integration of machine learning, structured user interfaces, and accessibility features ensures that the system is practical for end users.

8.2 Achievement of Objectives

The following describes the project's objectives from Chapter 1 were fulfilled with the implemented system:

1. To develop and train a machine learning model capable of predicting specific diseases, achieving a prediction accuracy of 85% or higher on the test dataset.
2. To design a user-friendly web application and evaluate its usability through User Acceptance Testing (UAT), ensuring that at least 90% of users rate its ease of use as 4 or higher on a 5-point Likert scale.

3. To design and test different prompts for large language model (Google Gemini), evaluating their effectiveness in advice generation and validate the outputs against trusted medical sources.

The objective 1 was successfully achieved by implemented a machine learning model trained on a structured dataset of symptoms and disease mappings. Several models including Random Forest, Decision Tree, and Support Vector Machine were evaluated on the validation dataset. Random Forest model demonstrated the highest accuracy and was therefore selected for final use. The final trained model achieved an accuracy of approximately 97% on the test dataset, exceeding the targeted threshold of 85%, thereby validating its reliability in predicting potential diseases based on user inputs. The final trained model was integrated into the system.

For objective 2, the frontend of the system was developed using React with an emphasis on intuitive navigation, clear visual design and responsiveness. To evaluate its usability, a User Acceptance Test (UAT) was conducted using a Google Form survey with a 5-point Likert scale. The results indicated that more than 90% of participants rated the system ease of use as 4 or above, thus meeting the target benchmark. This confirms that the web application successfully achieved its goal of providing a user-friendly interface suitable for both technical and non-technical users.

The objective 3 was addressed by conducting a prompt design study for the Google Gemini large language model. Several prompting strategies, including zero-shot prompt, role-based prompting, and chain-of-though prompt were evaluated in terms of accuracy, clarity, completeness and response time. The role-based prompting was found to deliver the most balanced outputs, ensuring medically relevant advice while maintaining consistency with establish guidelines. The generated advice was compared against trusted medical resource such as WHO and MedlinePlus, to validate the accuracy and reliability.

In summary, all major objectives outlined in Chapter 1 were successfully fulfilled. The machine learning model achieved the expected accuracy level. The user interface confirmed its user-friendliness through User Acceptance Testing (UAT) results. Combined with Google Gemini's prompt engineering technology, it effectively implemented recommendation generation, validated by reliable reference materials. These achievements collectively demonstrate that the project successfully met the established objectives.

8.3 Limitations and Recommendations for Future Works

Although the system operates effectively, several limitation and improvements or extensions can be made in future development.

Limitation	**Recommendations for Future Works**
Not replace the medical professional	Integrating the system with healthcare providers to ensure the predictions and recommendations are medically validated, ensuring enhancing trust and reliability. Furthermore, enhance Natural Language Processing (NLP) capability to handle complex sentences structures and diverse user input.
Not cover rare or new disease	By incorporating clinical trial data, the latest medical literature, and real-time medical databases to expand the dataset, which would allow the system to adapt to emerging health challenges.
Accuracy depends on the quality and quantity of dataset	Building larger and more diverse datasets, ideally sourced from multiple healthcare settings, to enhance the robustness and generalizability of predictions results. Integrate trusted medical databases or APIs such as Infermedia or WHO to provide more comprehensive advice and add severity-based guidance.
External factors	Designing an adaptive system that can integrate rapidly public health data during such scenarios, enabling the model to

	respond more effectively to novel conditions. Besides that, can implement offline functionality for areas with limited internet access.
Only supports English input and output	Implement multi-language support for non-English users, making the system more inclusive and suitable for a global user base.

By implementing these recommendation and future works, the system can gradually evolve into a more precise, reliable, and widely adopted medical support tool, making significant contributions to early disease warning and preventive healthcare practices.

REFERENCES

Ahmed, N et al. (2021) 'Machine learning based diabetes prediction and development of smart web application', *International Journal of Cognitive Computing in Engineering*, 2, pp. 229–241. Available at: https://doi.org/10.1016/j.ijcce.2021.12.001 (Accessed: 29 April 2025).

Alahmar, M. et al. (2023) 'Naïve Bayes Algorithms', *ResearchGate* [Preprint]. Available at: https://doi.org/10.13140/RG.2.2.15378.73921 (Accessed: 29 April 2025).

Ansarullah, S.I. et al. (2022) 'Significance of visible non-invasive risk attributes for the initial prediction of heart disease using different machine learning techniques', *Computational Intelligence and Neuroscience*, 2022, 9580896. Available at: https://doi.org/10.1155/2022/9580896 (Accessed: 29 April 2025).

Ashfakul Karim Kausik et al. (2025) 'Machine learning algorithms for manufacturing quality assurance: a systematic review of performance metrics and applications', *Array*, pp. 100393. Available at: https://doi.org/10.1016/j.array.2025.100393 (Accessed: 29 April 2025).

Bansal, M., Goyal, A. and Choudhary, A. (2022) 'A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning', *Decision Analytics Journal*, 3, 100071. Available at: https://doi.org/10.1016/j.dajour.2022.100071 (Accessed: 29 April 2025).

Berman, J. (2022) 'Understanding the DASH diet: MedlinePlus Medical Encyclopedia', *MedlinePlus*. Available at: https://medlineplus.gov/ency/patientinstructions/000784.htm (Accessed: 5 September 2025).

BERNAMA (2024) 'Long queues at public hospitals due to high patient load, staff shortages', 19 January. Available at: https://bernama.com/en/news.php?id=2389323 (Accessed: 29 April 2025).

Blockeel, H. et al. (2023) 'Decision trees: from efficient prediction to responsible AI', *Frontiers in Artificial Intelligence*, 6, 1115069. Available at: https://doi.org/10.3389/frai.2023.1115069 (Accessed: 29 April 2025).

CDC (2024) 'Measure your blood pressure', *High Blood Pressure*. Available at: https://www.cdc.gov/high-blood-pressure/measure/index.html (Accessed: 5 September 2025).

Cervantes, J. et al. (2020) ‘A comprehensive survey on support vector machine classification: applications, challenges and trends’, *Neurocomputing*, 408, pp. 189–215. Available at: https://doi.org/10.1016/j.neucom.2019.10.118 (Accessed: 29 April 2025).

Cleveland Clinic (2023) ‘Blood sugar monitoring: Why, how & when to check’, *Cleveland Clinic*. Available at: https://my.clevelandclinic.org/health/treatments/17956-blood-sugar-monitoring (Accessed: 5 September 2025).

Cleveland Clinic (2025) ‘Chickenpox’, *Cleveland Clinic.* Available at: https://my.clevelandclinic.org/health/diseases/4017-chickenpox (Accessed: 6 September 2025).

Cleveland Clinic (2025) ‘Ocular migraine’, *Cleveland Clinic.* Available at: https://my.clevelandclinic.org/health/diseases/24961-ocular-migraine (Accessed: 6 September 2025).

Gadesha, V. (2025) ‘Prompt engineering techniques’, *IBM*, 14 July. Available at: https://www.ibm.com/think/topics/prompt-engineering-techniques (Accessed: 5 September 2025).

Gadesha, V., Kavlakoglu, E. and Winland, V. (2025) ‘Chain of thoughts’, *IBM*, 14 July. Available at: https://www.ibm.com/think/topics/chain-of-thoughts (Accessed: 5 September 2025).

GeeksforGeeks (2020) *Differences between Django vs Flask*. Available at: https://www.geeksforgeeks.org/differences-between-django-vs-flask/ (Accessed: 29 April 2025).

GeeksforGeeks (2023a3) *Flask tutorial*. Available at: https://www.geeksforgeeks.org/flask-tutorial/ (Accessed: 29 April 2025).

GeeksforGeeks (2023b) *Top front-end frameworks in 2023*. Available at: https://www.geeksforgeeks.org/top-front-end-frameworks/ (Accessed: 29 April 2025).

GeeksforGeeks (2025) ‘Role-based prompting’, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/artificial-intelligence/role-based-prompting/ (Accessed: 5 September 2025).

Centers for Disease Control and Prevention (2024) Clinical features of malaria. Available at: https://www.cdc.gov/malaria/hcp/clinical-features/index.html/ (Accessed: 6 September 2025).

George, A.R. et al. (2024) ‘Multiple disease prediction using machine learning with chatbot and doctor-patient appointment system’, *2024 International Conference on Sustainable Power and Control*

Renewable Energies (ICSPCRE), pp. 1–6. Available at: https://doi.org/10.1109/icspcre62303.2024.10674954 (Accessed: 29 April 2025).

Gilbert, S. et al. (2020) 'How accurate are digital symptom assessment apps for suggesting conditions and urgency advice? A clinical vignettes comparison to GPs', *BMJ Open*, 10(12), e040269. Available at: https://doi.org/10.1136/bmjopen-2020-040269 (Accessed: 29 April 2025).

Gomathy, C. and Naidu, M.A.R. (2021) 'The prediction of disease using machine learning', *International Journal of Scientific Research in Engineering and Management (IJSREM)*, 5(10), pp. 1–7. Available at: https://doi.org/10.55083/ijsrem.2021.v05i10.001 (Accessed: 29 April 2025).

Gupta, S., Pal, K. and Choudhury, D. (2024) 'An experimental analysis of multiple disease prediction using machine learning algorithms', in *2024 International Conference on Computing, Information, and Networks (CICN)*, pp. 200–206. Available at: https://doi.org/10.1109/cicn63059.2024.10847453 (Accessed: 29 April 2025).

Harish Rajora et al. (2021) 'Web based disease prediction and recommender system', *arXiv* [Preprint]. Available at: https://doi.org/10.48550/arXiv.2106.02813 (Accessed: 29 April 2025).

Hello Doktor (2017) 'Do alcohol and cigarettes lead to hypertension', *Hello Doktor*. Available at: https://hellodoktor.com/en/alcohol-cigarettes-lead-hypertension/ (Accessed: 5 September 2025).

Hossain, M.I. (2023) 'Software Development Life Cycle (SDLC) methodologies for information systems project management', *International Journal For Multidisciplinary Research*, 5(5), e6223. Available at: https://doi.org/10.36948/ijfmr.2023.v05i05.6223 (Accessed: 29 April 2025).

Jain, A. (2024) 'SVM kernels and its type', *Medium*, 11 September. Available at: https://medium.com/@abhishekjainindore24/svm-kernels-and-its-type-dfc3d5f2dcd8 (Accessed: 29 April 2025).

Jaiman, A. (2024) 'Prompt engineering', *Medium*. Available at: https://ashishjaiman.medium.com/prompt-engineering-quick-reference-7801a033823a (Accessed: 5 September 2025).

Kavanagh. et al. (2017). 'Estimating Hospital-Related Deaths Due to Medical Error.' *Journal of Patient Safety,* 13(1), pp.1–5. Available at: doi:https://doi.org/10.1097/pts.0000000000000364. (Accessed: 5 May 2025).

Kosarkar, N. et al. (2022) ‘Disease prediction using machine learning’, in *2022 10th International Conference on Emerging Trends in Engineering and Technology - Signal and Information Processing (ICETET-SIP-22),* Nagpur, India, pp. 1–4. Available at: https://doi.org/10.1109/ICETET-SIP-2254415.2022.9791739 (Accessed: 29 April 2025).

Matzavela, V. and Alepis, E. (2021) ‘Decision tree learning through a predictive model for student academic performance in intelligent M-Learning environments’, *Computers and Education: Artificial Intelligence*, 2, 100035. Available at: https://doi.org/10.1016/j.caeai.2021.100035 (Accessed: 29 April 2025).

Mayo Clinic (2024) ‘Diabetes and exercise: When to monitor your blood sugar’, *Mayo Clinic*. Available at: https://www.mayoclinic.org/diseases-conditions/diabetes/in-depth/diabetes-and-exercise/art-20045697 (Accessed: 5 September 2025).

Mayo Clinic (2025) ‘Type 2 diabetes - diagnosis and treatment’, *Mayo Clinic*. Available at: https://www.mayoclinic.org/diseases-conditions/type-2-diabetes/diagnosis-treatment/drc-20351199 (Accessed: 5 September 2025).

Mayo Clinic (2024). *Heart attack symptoms: Know emergency signs*. [online] Mayo Clinic. Available at: https://www.mayoclinic.org/diseases-conditions/heart-attack/in-depth/heart-attack-symptoms/art-20047744. (Accessed: 6 September 2025).

Mayo Clinic Staff (2025) ‘Exercise and stress: Get moving to manage stress’, *Mayo Clinic*. Available at: https://www.mayoclinic.org/healthy-lifestyle/stress-management/in-depth/exercise-and-stress/art-20044469 (Accessed: 5 September 2025).

Mayo Clinic (2023) ‘Migraine – symptoms & causes’, *Mayo Clinic*. Available at: https://www.mayoclinic.org/diseases-conditions/migraine-headache/symptoms-causes/syc-20360201 (Accessed: 6 September 2025).

MedlinePlus (2019) ‘Diabetic diet’, *MedlinePlus*. Available at: https://medlineplus.gov/diabeticdiet.html (Accessed: 5 September 2025).

Newman-Toker, D.E. et al. (2023) 'Burden of serious harms from diagnostic error in the USA,' BMJ Quality & Safety, 33(2), pp. 109–120. https://doi.org/10.1136/bmjqs-2021-014130 (Accessed: 29 April 2025).

NHS Choices (2020) ‘Medically unexplained symptoms’, *NHS*. Available at: https://www.nhs.uk/conditions/medically-unexplained-symptoms/ (Accessed: 5 September 2025).

Pajila, P.J.B. et al. (2023) 'A comprehensive survey on Naive Bayes algorithm: advantages, limitations and applications', in *2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS)*, pp. 1497–1502. Available at: https://doi.org/10.1109/icosec58147.2023.10276274 (Accessed: 29 April 2025).

Reynolds, A. and Mitri, J. (2024) 'Nutritional recommendations for individuals with diabetes', *National Institutes of Health (NIH)*. Available at: https://www.ncbi.nlm.nih.gov/books/NBK279012/ (Accessed: 5 September 2025).

Rural Health Information Hub (2024). 'Healthcare access in rural communities.' [online] Rural Health Information Hub. Available at: https://www.ruralhealthinfo.org/topics/healthcare-access. (Accessed: 5 May 2025).

Sangeetha, V. et al. (2024) 'Revolutionizing healthcare: screening system to identify diseases using machine learning approach', in *2024 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE)*, pp. 1–6. Available at: https://doi.org/10.1109/IITCEE59897.2024.10607516 (Accessed: 29 April 2025).

Saravanan, T. et al. (2020) 'Comparative analysis of software life cycle models', in *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 866–871. Available at: https://doi.org/10.1109/icacccn51052.2020.9362931 (Accessed: 29 April 2025).

Sheldon, R. (2023) 'What is web development framework (WDF)?', *SearchContentManagement* [online]. Available at: https://www.techtarget.com/searchcontentmanagement/definition/web-development-framework-WDF (Accessed: 29 April 2025).

Song, J. et al. (2021) 'The Random Forest model has the best accuracy among the four pressure ulcer prediction models using machine learning algorithms', *Risk Management and Healthcare Policy*, 14, pp. 1175–1187. Available at: https://doi.org/10.2147/rmhp.s297838 (Accessed: 29 April 2025).

Sreedevi, B. et al. (2022) 'Web based disease prediction and forecasting with KNN and RNN using Internet of Medical Things', in *2022 International Conference on Computer, Power and Communications (ICCPC)*, pp. 192–198. Available at: https://doi.org/10.1109/iccpc55978.2022.10072288 (Accessed: 29 April 2025).

Srihith, I.D. et al. (2023) 'A forest of possibilities: decision trees and beyond', *Journal of Advancement in Parallel Computing*, 6(3), pp. 29–37.

Available at: https://doi.org/10.5281/zenodo.8372196 (Accessed: 29 April 2025).

Taunk, K. et al. (2019) 'A brief review of nearest neighbor algorithm for learning and classification', in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pp. 1258–1263. Available at: https://doi.org/10.1109/ICCS45141.2019.9065747 (Accessed: 29 April 2025).

V. Sharmila *et al.* (2024) *Challenges in Information, Communication and Computing Technology*, *CRC Press eBooks*. Informa. Available at: https://doi.org/10.1201/9781003559085 (Accessed: 29 April 2025).

W3Schools (2020) 'React tutorial'. Available at: https://www.w3schools.com/react/default.asp (Accessed: 29 April 2025).

World Health Organization (2024) 'Diabetes', *World Health Organization*. Available at: https://www.who.int/news-room/fact-sheets/detail/diabetes (Accessed: 3 September 2025).

World Health Organization (2021) 'Healthy diet', *World Health Organization*. Available at: https://www.who.int/initiatives/behealthy/healthy-diet (Accessed: 5 September 2025).

World Health Organization (2023) 'Hypertension', *World Health Organization*. Available at: https://www.who.int/news-room/fact-sheets/detail/hypertension (Accessed: 3 September 2025).

World Health Organization (2024) 'Noncommunicable diseases', *WHO Fact Sheets* [online]. Available at: https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases (Accessed: 29 April 2025).

Zhu, T. (2020) 'Analysis on the applicability of the Random Forest', *Journal of Physics: Conference Series*, 1607(1), 012123. Available at: https://doi.org/10.1088/1742-6596/1607/1/012123 (Accessed: 29 April 2025)

APPENDICES

Appendix A: Questionnaire for User Acceptance Testing (UAT)

18/09/2025, 17:29 Disease Prediction Web Application using Machine Learning - User Acceptance Test (UAT) Form

Disease Prediction Web Application using Machine Learning - User Acceptance Test (UAT) Form

Thank you for participating in the User Acceptance Testing (UAT) for my Final Year Project titles "Disease Prediction Web Application using Machine Learning". This form should take approximately 3-5 minutes to complete. Your feedback is crucial to ensure the application is functional, reliable and overall satisfaction of the system. Please answer honestly based on your experience using the system. All responses will remain confidential and be used solely for improving the application.

If you have any inquires, please contact Foo Jia Yu, email: jiayu03@1utar.my.

* Indicates required question

1. Age *

   Mark only one oval.

   - ◯ 18-21
   - ◯ 22-30
   - ◯ 31-45
   - ◯ >45

2. Occupation/Education Level *

   ____

Section B: User Acceptance Test

For scale-based questions, select your answer below where:
1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree

https://docs.google.com/forms/d/1iq9lsGTZ8er6TZ5TeLWx-dZ1zSDTworyZI8I6vU1FtY/edit 1/5

3. 1. How easy was it to use and navigate the system? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Easy

4. 2. How easy was it to input symptoms into the system? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Easy

5. 3. How clear and appropriate were the feedback or error messages when incorrect or incomplete data was entered? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Clear/Appropriate

6. 4. The system successfully generated a disease prediction based on the input provided. *

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

7. 5. How accurate were the disease predictions provided by the system? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Accurate

8. 6. How relevant and useful was the medical advice provided after the prediction? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Relevant/Useful

9. 7. How accessible and helpful was the prediction history feature? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Accessible/Helpful

10. 8. How satisfied are you with the overall system?

Mark only one oval.

- ◯ Very satisfied
- ◯ Satisfied
- ◯ Neutral
- ◯ Dissatisfied
- ◯ Very dissatisfied

18/09/2025, 17:29 Disease Prediction Web Application using Machine Learning - User Acceptance Test (UAT) Form

11. 9. Did not encounter major difficulties while using the application.

Mark only one oval.

	1	2	3	4	5	
Stro	◯	◯	◯	◯	◯	Strongly Agree

12. 10. What suggestions do you have to improve the system?

Mark only one oval.

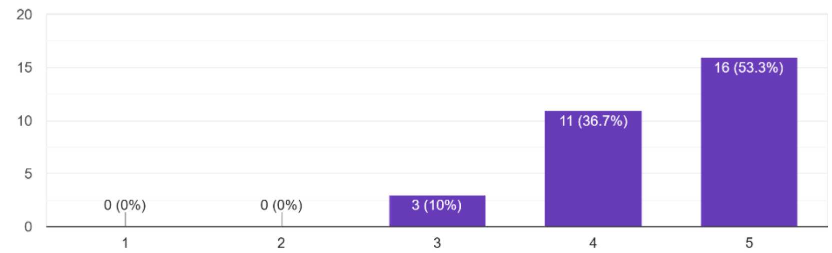
◯ No

◯ Other: ____________________

Appendix B: Results for User Acceptance Testing (UAT)

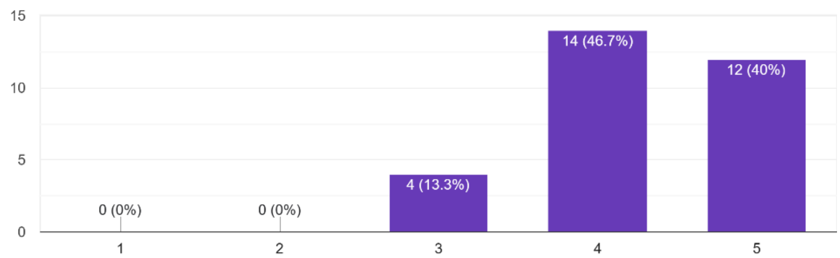
1. How easy was it to use and navigate the system?

30 responses



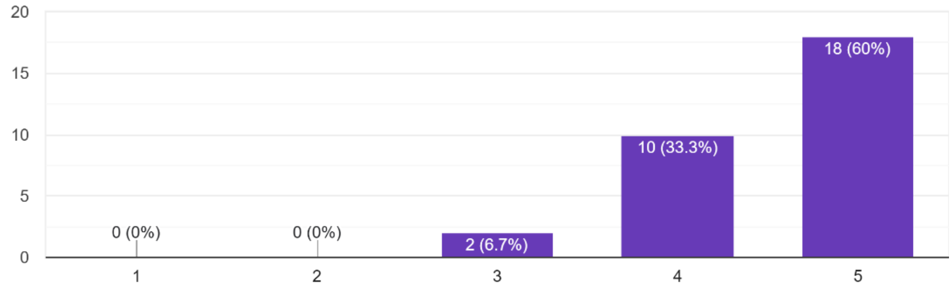
2. How easy was it to input symptoms into the system?

30 responses



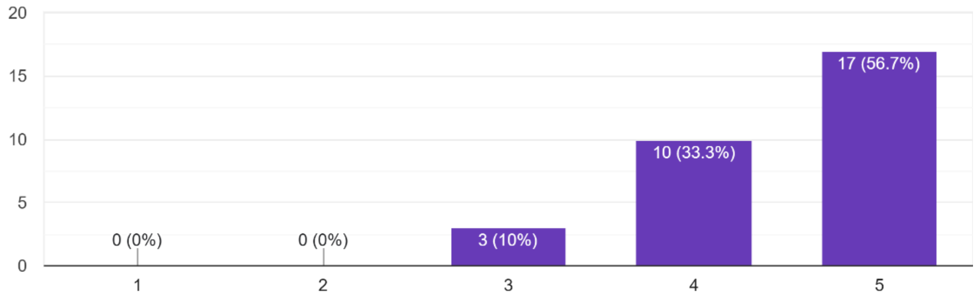
3. How clear and appropriate were the feedback or error messages when incorrect or incomplete data was entered?

30 responses



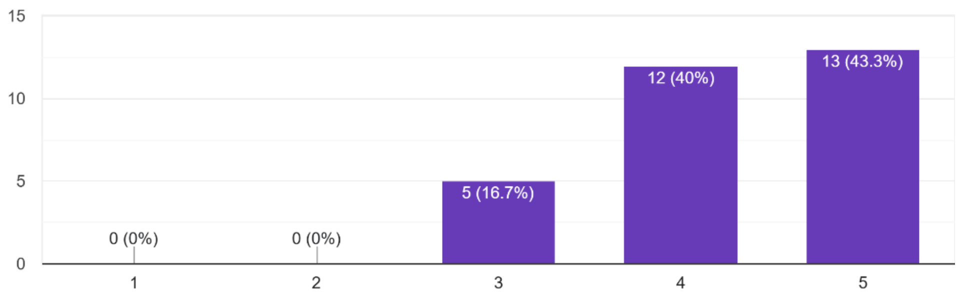
4. The system successfully generated a disease prediction based on the input provided.

30 responses

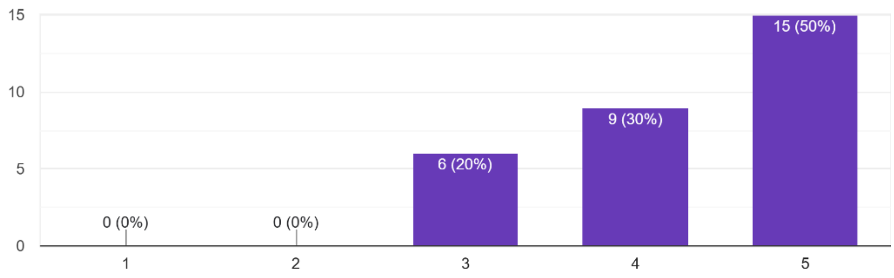


5. How accurate were the disease predictions provided by the system?

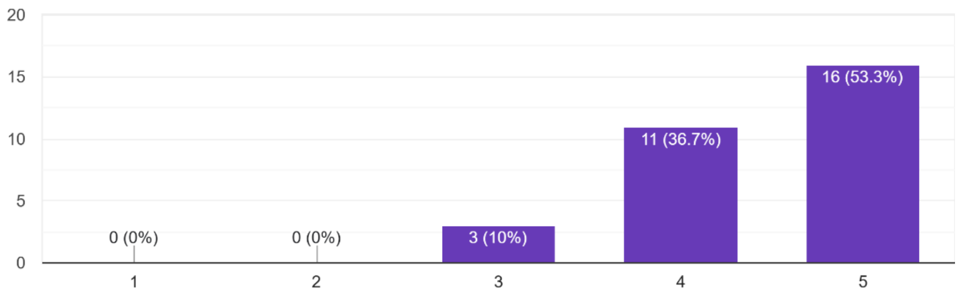
30 responses



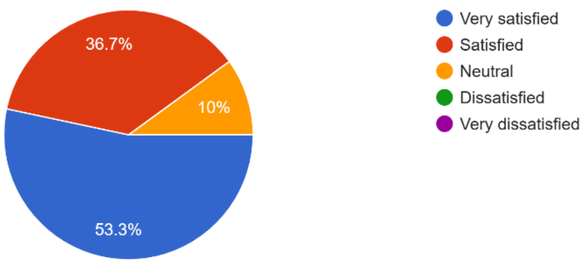
6. How relevant and useful was the medical advice provided after the prediction?
30 responses



7. How accessible and helpful was the prediction history feature?
30 responses

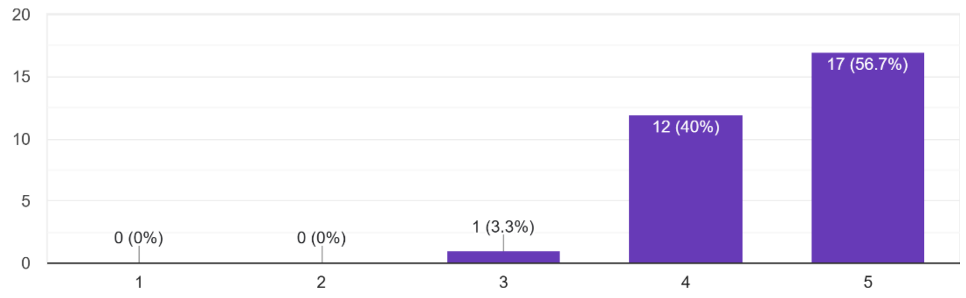


8. How satisfied are you with the overall system?
30 responses



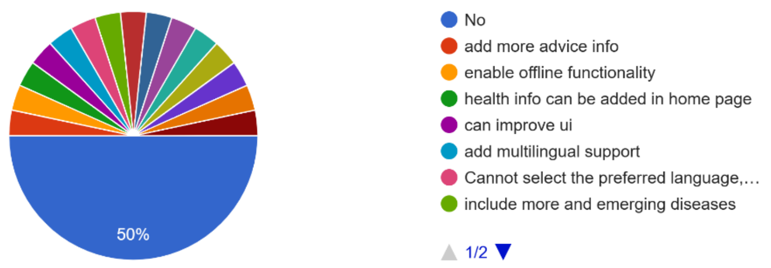
9. Did not encounter major difficulties while using the application.

30 responses



10. What suggestions do you have to improve the system?

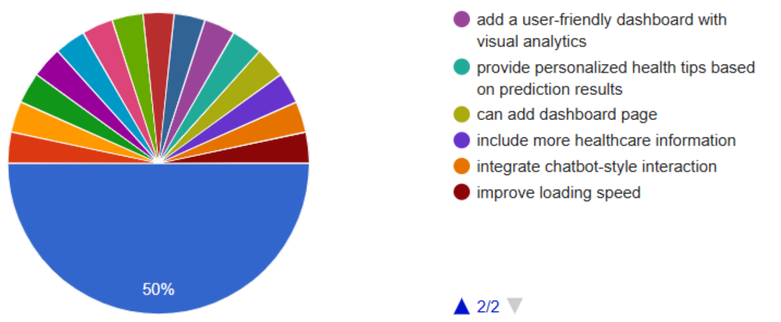
30 responses



10. What suggestions do you have to improve the system?

Copy chart

30 responses



Appendix C: Questionnaire for User Interface (UI) Design Feedback

Disease Prediction Web Application using Machine Learning - User Interface Design Feedback Form

Thank you for participating in the User Interface Design Feedback for my Final Year Project titles "Disease Prediction Web Application using Machine Learning". This form should take approximately 3-5 minutes to complete. Your feedback is essential to refining the system's design and ensuring it is visually appealing, intuitive, and accessible. Please answer honestly based on your experience using the system. All responses will remain confidential and be used solely for improving the application.

If you have any inquires, please contact Foo Jia Yu, email: jiayu03@1utar.my.

* Indicates required question

1. Age *

   Mark only one oval.

   - 18-21
   - 22-30
   - 31-45
   - >45

2. Occupation/Education Level *

   ____

Section B: Interface Design

For scale-based questions, select your answer below where:
1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree

3. 1. How easy was it to use the system without prior training? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Easy

4. 2. How intuitive was the user interface (UI) design? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Intuitive

5. 3. How visually appealing is the overall design of the system (e.g., colours, fonts, layout)? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Appealing

6. 4. How easy on the eyes and consistent with the theme are the colours used in the system? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Easy/Consistent

7. 5. How clear and descriptive were the menu labels and buttons? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Clear

8. 6. How easy was it to navigate through the system's menus and pages? *

Mark only one oval.

	1	2	3	4	5	
Very	◯	◯	◯	◯	◯	Very Easy

9. 7. How responsive was the application (e.g., button clicks, loading times)? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Responsive

10. 8. How clear were the feedback or error messages? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Clear

11. 9. How confident do you feel using the UI for disease prediction tasks? *

Mark only one oval.

	1	2	3	4	5	
Not	◯	◯	◯	◯	◯	Very Confident

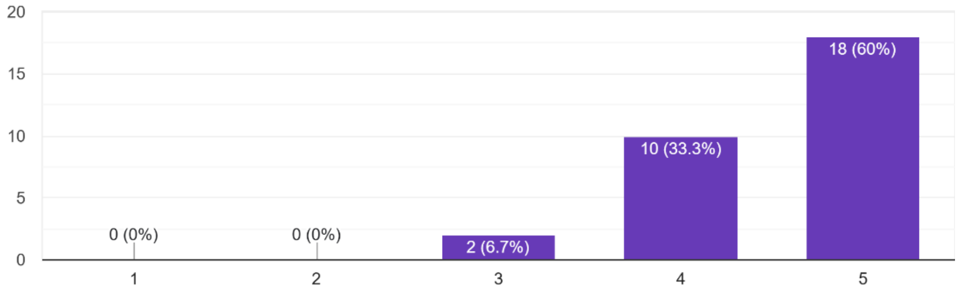
12. 10. What UI improvements would you suggest?

Please share any suggestions to enhance the UI's usability, design, or functionality. If none, you may leave this blank.

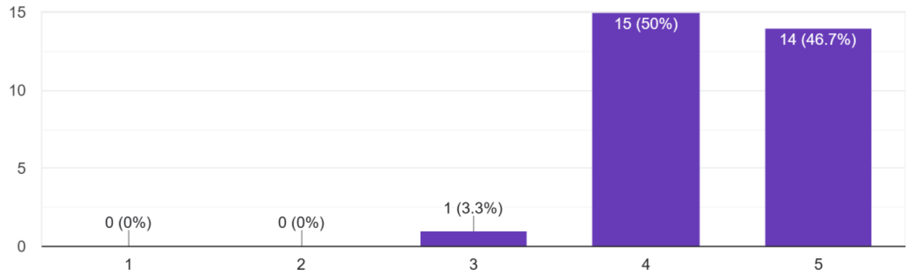
Appendix D: Results for User Interface Design Feedback

1. How easy was it to use the system without prior training?

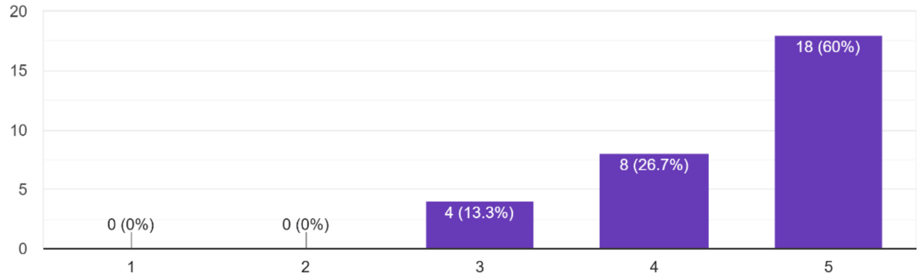
30 responses



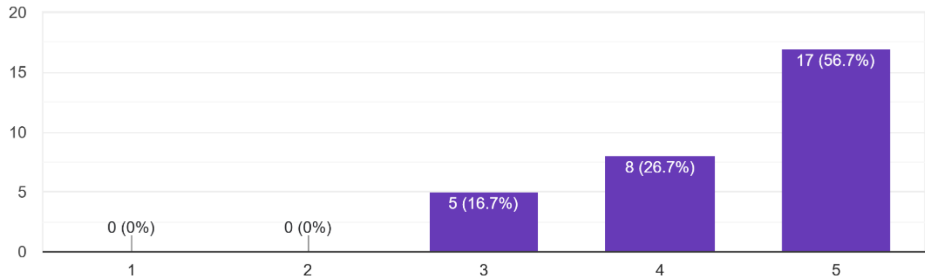
2. How intuitive was the user interface (UI) design?
30 responses



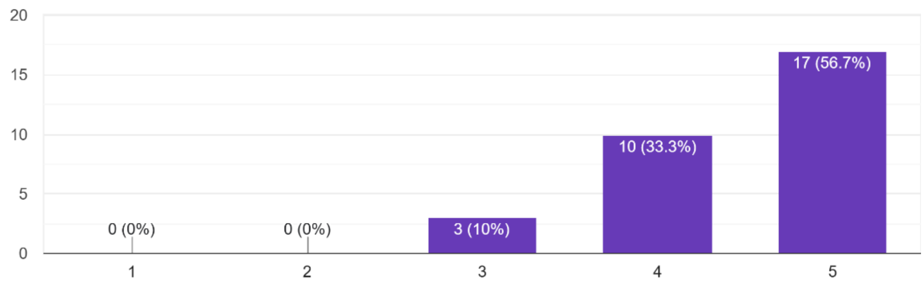
3. How visually appealing is the overall design of the system (e.g., colours, fonts, layout)?
30 responses



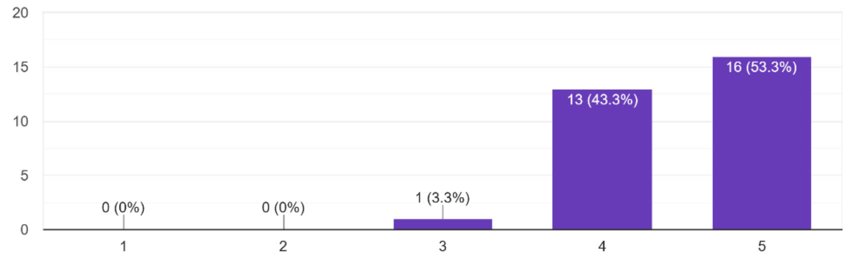
4. How easy on the eyes and consistent with the theme are the colours used in the system?
30 responses



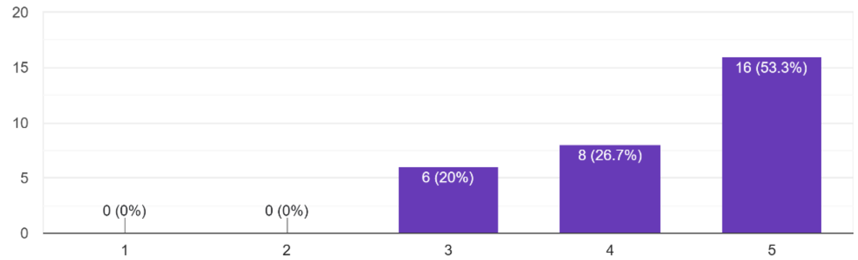
5. How clear and descriptive were the menu labels and buttons?
30 responses



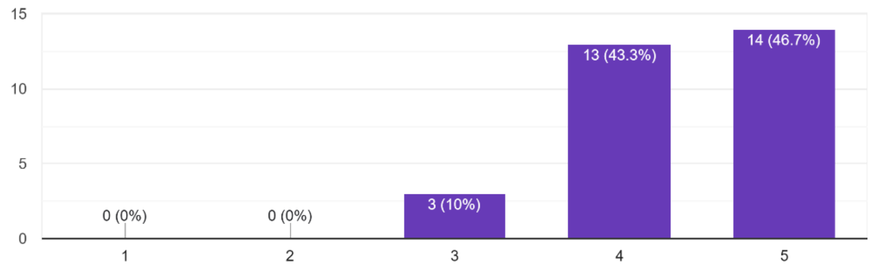
6. How easy was it to navigate through the system's menus and pages?
30 responses



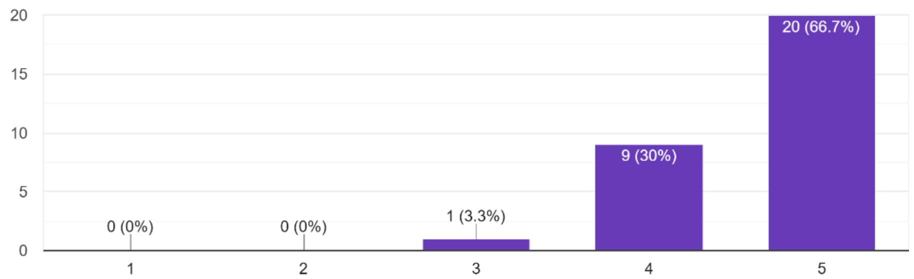
7. How responsive was the application (e.g., button clicks, loading times)?
30 responses



8. How clear were the feedback or error messages?
30 responses



9. How confident do you feel using the UI for disease prediction tasks?
30 responses



10. What UI improvements would you suggest?

11 responses

- can add dark mode for better accessibility
- provide a dashboard with quick access features
- can have more dynamic home page
- Add tooltips or hints for new users
- Use a cleaner, more modern layout
- Add progress indicators when loading results
- use more visuals (charts, graphs) for results
- make buttons larger and easier to tap
- Improve spacing and alignment of elements
- Improve color contrast for better readability
- Personalize the interface