Disease Prediction System

A PROJECT REPORT
for
Project (KCA451)
Session (2024-25)

Submitted by

SHIVANK NARUKA
(2300290140174)
UTKARSH KUMAR SINGH
(2300290140198)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Ritesh Kumar Gupta
Assistant Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(APRIL 2025)

CERTIFICATE

Certified that **Shivank Naruka (2300290140174), Utkarsh Kumar Singh (2300290140198)** has/ have carried out the project work having "**Disease Prediction System**" (**Project-KCA451**) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

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

Date:

Mr. Ritesh Kumar Gupta
Assistant Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Dr. Akash Rajak
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Disease Prediction System

ABSTRACT

The swift development of Artificial Intelligence (AI) and Machine Learning (ML) has transformed numerous sectors, with healthcare being one of the most vital domains of innovation. This project, "Smart Disease Predictor", offers a web-based application that can predict likely diseases from user-input symptoms using trained machine learning models. The system is built with the Python Flask framework and incorporates main ML algorithms like Naive Bayes, Random Forest, and Logistic Regression to predict and classify diseases with a significant level of accuracy.The application enables users to input their symptoms via a basic web interface and get a forecasted disease as an output. The backend compares the input against a trained dataset of binary values for symptoms against different diseases. Of several trained models, the highest performing one is chosen and implemented in real-time for prediction. This project also displays disease frequency patterns and tests model performance using accuracy measures.The system is mainly designed to be a front-end tool that generates early awareness and encourages users to seek professional medical help if needed. Not a substitute for certified medical diagnosis, this intelligent system can prove to be very useful in remote locations, emergency situations, or as a self-checking device. The project finishes by proposing future improvements such as the addition of real-world data sets, wearable device integration, and multi-language interfaces to make the tool stronger and more accessible.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Mr. Ritesh Kumar Gupta** for his guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak**, Professor and Dean, Department of Computer Applications, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Shivank Naruka

Utkarsh Kumar Singh

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The Smart Disease Predictor is an intelligent, user-centric web-based system developed to assist users in predicting diseases based on the symptoms they report. It serves as a preliminary diagnostic tool, powered by machine learning, which takes user-inputted symptoms and identifies the most probable disease. By leveraging advanced classification models such as Naive Bayes, Random Forest, and Logistic Regression, the system can provide fast, reasonably accurate, and informative predictions. This ensures that users can make informed decisions on seeking further medical attention.

What makes this system even more compelling is its integrated chatbot that enhances the interactivity and accessibility of the application. Instead of navigating through multiple pages or interfaces, users can simply engage in a chat with the bot to clarify doubts, receive guidance, and understand predictions in a conversational format.

The application is designed using Python's Flask web framework, making it modular, scalable, and easy to deploy. It reads a binary symptom-disease dataset, trains multiple models, evaluates them for accuracy, selects the best-performing model, and uses it in the live web application.

1.2 BACKGROUND

Healthcare is still among the most important pillars of human development. Even with dramatic progress being made in medical science and healthcare infrastructure, millions globally continue to have no access to timely and proper medical advice. This vacuum becomes extremely acute in rural or unserved regions, in times of epidemic or pandemic, and in crises when expert advice is not readily on health.

Artificial Intelligence (AI), specifically in the health sector, has thrown open new avenues. AI-based diagnostics, symptom checkers, and predictive models are becoming more and more widely used to fill the gap in diagnostics. But most such systems are non-personalized, non-user-friendly, or too technical for common people.

The Smart Disease Predictor seeks to address these problems. It combines AI and chat UI in one application, enabling users to communicate naturally with the system, receive disease predictions, and learn the reasoning behind them — all from the comfort of their own devices.

The first healthcare applications of AI were rule-based expert systems. These systems emulated the decision-making capacity of medical experts using pre-established rules. They were, however, inflexible, non-adaptive, and could not generalize to situations beyond the ones they were specifically programmed for. With advances in machine learning and deep learning, these disadvantages have been addressed to a significant degree. ML-based models are learned using large datasets and can recognize complex and frequently non-linear relationships between input (symptoms) and output (diseases). These systems learn from the data, enabling generalization and prediction accuracy improvement over time.

Yet, the majority of AI-based health apps target individual diseases like diabetes, cancer, or heart disease. Few platforms are designed to offer general disease prediction from several symptoms. Even those that do exist, many do not provide real-time, interactive assistance or are inaccessible to non-experts because of difficult interfaces. Additionally, most online symptom checkers lack transparency in their decision-making process, leaving users perplexed or worried about the outcome they get.

This project is conceived as an answer to such gaps. It offers a web-based, simple-to-use platform where users may input their symptoms in natural language and obtain a predicted diagnosis by well-trained ML models. Better still, it incorporates an interactive chatbot through which dialogue can be conducted with the system. Users can pose clarifying questions, request further information, and interact with the system as if in a human dialogue. This incorporation makes users, particularly those who are not experienced in medical or technical terms, able to comprehend their outcome and take the necessary course of action.

The impetus for this project also involves the COVID-19 pandemic, which revealed major weaknesses in global health infrastructures. In lockdowns and shortages of healthcare resources, many people were denied proper diagnosis or medical care. Under such circumstances, self-diagnostic applications or AI-based triage systems became critical. This further underlined the role of digital healthcare technologies, not only as auxiliary tools but as frontline support systems during emergencies.

Educationally, the project also demonstrates how machine learning, web development, natural language processing, and user experience design can be combined to address real-world challenges. For computer science students and developers, it provides a practical demonstration of using Python, Flask, and classification algorithms

such as Naive Bayes and Random Forest. It also demonstrates the importance of front-end design in making AI models usable and effective.

From a societal standpoint, bringing healthcare tools into the people empowers them. When individuals can have good predictions and information on health at their fingertips, they become more health-conscious. Early detection, even by AI-driven estimates, can be a revolution in keeping hospital loads light, stopping disease progress, and ensuring better overall health.

1.3 OBJECTIVES

1. To Create a Symptom-Based Disease Prediction Model

The core of the Smart Disease Predictor is a machine learning model that can effectively classify diseases from input symptoms. The model must be trained on a well-organized, pre-labelled dataset where symptoms are encoded as features and diseases as output classes. Several models (Naive Bayes, Random Forest, Logistic Regression) are tested and compared to choose the most efficient and accurate algorithm for prediction.

Key Points:

- Transform raw data into a machine-readable data structure (feature matrix).
- Preprocess the data, normalize, and encode.
- Train and test the models for performance metrics like accuracy, precision, and recall.
- Save the best-performing model with the help of joblib for live predictions.

2. To Create a User-Friendly Web Application

The system should have a frontend through which users can input symptoms in an organized and simple way. Users should not need to be medical professionals to use the application. The web application is designed with the Flask framework in Python so that it can effectively communicate with the backend model and provide predictions in real-time.

Key Points:

- Develop a responsive UI using HTML, CSS, and JavaScript.
- Use Flask for handling requests, routing, and integration with the ML model.
- Display results clearly and in an understandable format.
- Ensure mobile and cross-browser compatibility.

3. To Encourage Preventive Healthcare by Early Diagnosis

Another key goal is to encourage preventive healthcare by prompting users to take action on symptoms before they become serious. The Smart Disease Predictor gives users timely feedback, enabling them to determine if their symptoms need to be addressed medically.

Key Points:

- Encourage users to be more health-aware through awareness.
- Enable detection of patterns in repeat symptoms.
- Enable users where there is poor access to healthcare professionals.

4. To Facilitate Modular and Scalable Architecture.

The structure and codebase of the application should be modular to ensure that improvements in the future (such as API integration, user authentication, or symptom validation) can be introduced without rewriting the whole system.
Key Points:

- Clean separation of frontend, backend, and ML logic.
- Upgradable components with ease.
- Clean and readable code to facilitate team development and versioning.

5. To Educate Users regarding Frequent Diseases and Symptoms

Aside from disease forecasting, the chatbot or application can be made to inform users about symptoms, reasons, and prevention regarding a variety of diseases.
Key Points:

- Function as a tool for health education.
- Deliver generic advice and information when suitable.
- Provide external medical references that are vetted (future potential).

1.4 APPLICABILITY

1. In Educational Institutions

This system may be utilized in educational institutions, particularly in the areas of computer science, biomedical engineering, and public health, as an instrument for teaching and learning. It may be utilized as a hands-on project to illustrate the real-world applications of artificial intelligence to healthcare.
Use Cases:

- Students of computer science studying machine learning algorithms.
- Medical students learning symptom-disease correlations.
- Health analytics for data science projects and research studies.
- Coding contests and hackathons where health apps need to be prototyped quickly.

2. For Clinics and Healthcare Providers

Hospitals and clinics, particularly those with high volumes of walk-in patients, would find it useful to have a system that does an initial triage by symptoms. As much as it is not a substitute for expert diagnosis, the Smart Disease Predictor can help healthcare practitioners by getting initial information prior to the patient's consultation with the physician.
Use Cases:

- Utilized by nurses or receptionists to enter symptoms of patients upon check-in.
- Clinic support tool to enable doctors shortlist potential conditions quickly prior to tests.
- Remote consultation support system for telemedicine or e-clinics.

3. For Government Health Programs and NGOs

Governments and non-governmental organizations that conduct health awareness drives or provide medical support in rural areas can utilize this system for mass screenings or educational purposes. Especially in underdeveloped or remote regions, the Smart Disease Predictor can serve as a lightweight diagnostic assistant.

Use Cases:

- Integrated into mobile health vans and pop-up clinics.
- Used in mass screening camps to identify people with potential illnesses.
- Deployed via smartphones in villages with the help of local health volunteers.

4. In Corporate Wellness Programs

Many companies now invest in employee wellness and mental health programs. The Smart Disease Predictor can be part of such initiatives to help employees identify early signs of illness and take preventive measures, contributing to better productivity and lower absenteeism.

Use Cases:

- Embedded into internal HR or employee assistance portals.
- Provided as a wellness feature in corporate health check-up drives.
- Used by company doctors during health assessments.

1.5 SCOPE

The Smart Disease Predictor project is aimed at creating a web application that predicts disease from user-symptom reported data using machine learning algorithms such as Naive Bayes, Random Forest, and Logistic Regression. The project scope involves data handling, training, evaluation, and selection of the optimal model for deployment. The backend of the project is developed using the Flask framework for handling user input and providing correct disease predictions. On the frontend, an intuitive interface is implemented using HTML and CSS to provide a seamless and accessible user experience. The project is restricted to disease prediction based on a predetermined set of symptoms from the dataset and does not involve real-time data updates, medical validation outside the dataset, or sophisticated natural language processing. Future development could include user authentication, tracking of medical history, and integration with real-world healthcare APIs.

1.6 ARCHITECTURE

Data Preparation and Collection:

- The dataset employed is a binary format of different symptoms for different diseases.
- A disease is represented by each row, and each column represents a symptom by 1 indicating its presence and 0 indicating absence.
- Data is preprocessed to provide clean input to machine learning algorithms, missing no values or being inconsistent.

Feature and Label Extraction:

- Features (X): Symptom columns are all taken as the input features.
- Labels (y): The target output is the first column, diseases.

Train-Test Split:

- Train and test sets are split from the dataset in the ratio of 80:20.
- This makes sure that the model is tested on unseen data to assess its actual prediction ability in real-world scenarios.

Model Building:

- Naive Bayes Classifier – most appropriate for categorical input data such as symptoms.
- Random Forest Classifier – ensemble learning method involving multiple decision trees.
- Logistic Regression – statistical model that is handy for binary and multi-class classification.

CHAPTER 2

FEASIBILITY STUDY

2.1 TECHNICAL FEASIBILITY

The technical feasibility of a project technical feasibility is the evaluation of the technical resources that are available to develop and implement the project. It will analyse whether the current technical environment can accommodate the requirements of the system and whether the proposed system can be developed with the existing technology. The technical feasibility is a key component in this case (in the context of our health based web application project) for this to be the case the system can be implemented with currently available hardware, software and technological expertise.

The proposed health insights and prediction system leverages technologies such as Flask (a Python-based web framework), HTML, CSS, JavaScript, SQLite, and various health-related datasets for prediction logic. Flask's suitability for developing small to medium sized web applications as well as its lightweight nature is the reason for the decision to use it. It is a simple way to route, template, and integrate with the database. Moreover, extensions that provide authentication, session management, and database interaction are supported by Flask, which are all critical components of our system.

In terms of hardware, requirements are especially minimal. The system is targeted to run on basic personal computers or laptops and the server load is not expected to be intense during development or initial deployment. Platforms such as Heroku, PythonAnywhere or even local deployment using a WSGI server (Gunicorn) is enough for hosting purpose. This means that users with standard hardware are able to contribute to the development, testing, and deployment processes without special or high end hardware.

They are also relatively lightweight as far as software requirements are concerned. The only thing I'd include in the development environment would be Python, Flask and a code editor like VS Code or PyCharm. All members of the development team can use these tools as they are open source or have a free community edition. Furthermore, HTML, CSS, and JavaScript are used in order to allow the frontend to be built and tested on any modern web browser without requiring any proprietary software. It also helps greatly with the technical feasibility since it eliminates the cost and complexity associated with a licensed tool.

SQLite is used from a database perspective since it is a lightweight, serverless database, requires minimal configuration, and comes bundled with Python. With this choice, data management for the development phase becomes easier and setup and testing becomes fast. The system is robust enough to allow a switch to more robust database systems such as PostgreSQL, MySQL, etc. in future if needed. In the current scope and scale of the project, SQLite has all the features required for user management, storing health predictions, and maintaining session data.

As for the technical feasibility aspect, it is also important to integrate together predictive algorithms based on user input. The application has health prediction functionalities based on logical conditions or machine learning models. Per the models, they have to be properly trained and tested to be accurate and relevant. In this context, the technical environment consists of using Python libraries such as Pandas, NumPy, and Scikit-learn that are widely used for data processing and machine learning. The fact that these libraries are open source and are easily integrated into Flask applications makes it technically feasible.

The application uses HTML5, CSS3 and JavaScript to develop the user interface, and to provide interactivity and a better user experience. These technologies have been chosen in such a way that they are cross browser compatible and responsive. Also, it is possible to integrate CSS frameworks like Bootstrap or utility first CSS libraries like Tailwind CSS to speed up UI development and keep your UI responsive. This ensures that users won't have performance problems while using the project from different devices, thereby enhancing the project's technical feasibility. The technical feasibility study also takes in to account security as a fundamental aspect. The system has basic security baked in, such as secure session management and password hashing, using Werkzeug or Bcrypt, and form validation that prevents SQL injection and cross site scripting attacks. The current implementation provides a basic level of protection, however the system architecture is designed to support future enhancements such as the deployment of HTTPS as well as role based access control and audit logging.

Scalability is another vital consideration. The system is designed to scale as well, although the first deployment is expected to serve a relatively small number of users. The code structure is modular, using MVC (Model View Controller) architecture as well as restful principles, which make it easier for adding additional features without much

disruption. When traffic grows, the system can then be migrated to more powerful hosting environments or cloud platforms such as AWS or Google Cloud. It guarantees that the technical framework will be able to support long term growth and extra functionalities.

As for team capability, the development team has the technical knowledge of Python, web development and database management. This is a positive aspect of the feasibility assessment as it takes away the requirement of need of an external consultant or training. The existing skill set allows for addressing technical challenges effectively and building the system in the projected timeline.

The Development process includes version control using Git so that the code can be managed efficiently and collaborated fine to roll back in case of an error. Platforms like GitHub or GitLab require no presence as they allow you to collaborate and create continuous integration/deployment pipelines if necessary. Such methodologies are in line with modern software development standards and improve the technical feasibility of the project.

To ensure smooth development, it is also vital to have vast documentation and community support on the chosen technologies (Flask, SQLite, HTML, CSS, JavaScript, and Python libraries). Online sources, forums, and tutorials as well as official documentation provide solutions to many problems and some guidelines on best practices. This abundant resource thus eliminates the effect of technical stagnation or delays on the entire project.

Lastly, a flexible technical stack makes allowances for subsequent upgrades or adaptations. Should any future changes occur in any of the machine learning models or new health data sets becomes made available, the system can simply be updated without lengthy complete redesigns. Additional modules can also be created and plugged into the system without the need for overall modifications at times when the requirements of users change.

2.2 Economic Feasibility

An economic feasibility study is perhaps one of the most critical components in determining whether or not a particular system is financially sustainable and technologically feasible over the long term. The economic feasibility study is able to highlight possible benefits from the financial viewpoint, and whether those benefits exceed the cost of the project. This section will focus on the costs, returns, budget estimates, breakeven, and the long-run sustainability of our health-based web application.

2.2.1 Current Cost Estimates

When developing a web application, there will be a number of financial commitments that need to be made at various stages. There will be initial costs that will relate to the initial setup of the infrastructure, licensing, software development, hosting, and your initial advertising/marketing. For the health-based application we propose, costs can be generally categorized and simplified in the following key areas:

- **Hardware and Infrastructure:** While the application is hosted in the cloud, the primary hardware required would be developer laptops and testing devices.

- **Software Tools and Licenses:** Development software such as IDEs (e.g., Visual Studio Code, PyCharm), design software (e.g., Figma, Adobe XD), and licenses for any third-party libraries or APIs.

- **Domain and Hosting Charges: Charges** for Domain name registration charge and cloud hosting services (e.g., AWS, Heroku or DigitalOcean).

- **Initial Development Team Payroll:** Hiring developers, UI/UX designers, and quality analysts during the initial build time.

- **Marketing & Launch:** Initial marketing to attract users which includes digital ads, SEO work, and various promotions.

The costs estimate of these for the first phase can approximate from INR 1,50,000 to 2,00,000 for a basic but scalable MVP (Minimum Viable Product).

2.2.2 Operational Costs

After the application is successfully deployed and online, the operational costs that we will incur while running and maintaining the application will be listed below:

- **Server & Cloud Maintenance:** Every month we will incur ongoing monthly charges for cloud services being used for hosting and databases.
- **Customer Support and Feedback Management:** Ongoing support via chatbots or through manual support staff.
- **Software Maintenance and Updates:** Regular bug fixes to resolve issues, updates and improvements based on user feedback.
- **Content Creation:** Health articles, newsletters and content moderation.
- **Salaries & Compensation for HR:** Pay for any full-time or part-time staff that will be managing operations in the background.

Ongoing operational costs will be between INR 20,000 and INR 30,000 per month depending on user load and features.

2.2.3 Cost-Benefits Analysis

Evaluating the economic viability of the application involves not only looking at the estimated cost, but also an estimated benefit. The cost-benefit analysis considers both tangible and intangible benefits.

- **Tangible Benefits:**
    - Human health consultation paid services.
    - Premium subscriptions (premium features: personalized diet charts, weekly medical reports, etc.)

    - Advertising revenue.
    - Revenue from affiliate marketing or partnerships with pharmacies and hospitals.
- **Intangible Benefits:**
    - Brand reputation and trustworthiness.
    - Positioning within the digital health tech space.
    - Long-term partnerships and funding.

Given the selection of monetization strategies and the current number of users, the application can expect to see a return on investment (ROI) within 12-18 months of launch.

2.2.4 Break-Even Analysis

The break-even point refers to the point at which total revenues equal total costs, and the business starts to enter the realm of profit. For our Web application, the point of the break-even analysis is to predict when we expect the investment will start to pay off.

Assumptions:

- Monthly costs = INR 25,000
- Average revenue/user/month= INR 150
- Active paying user needed = 167

If the marketing initiatives are maintained and there is a steady onboarding of users, reaching 167 paying users within 6 months is achievable. In this sense, we expect the application to break-even between 6–9 months of the application being launched.

2.2.5 Scalability and Financial Outlook over the Long-Term

This project's ability to scale is one of the biggest factors in determining its economic feasibility. We are building the application with a modular architecture so that it easily scales. As we add users and more features we expect to see the following:

Increase in Revenue: Each new premium feature offered will increase revenue per user, for example, AI diagnostics, video consultations, medical record integration.

Investor Interest: A scalable business model is attractive to angel investors and venture capitalists, reducing financial pressure going forward.

Economies of Scale: As we gain users the per-user cost to maintain will drop and allow for higher profit margins.

With a solid business case, it is possible that at the end of Year 2, the application could achieve annual profits of INR 5–7 lakhs.

2.2.6 Alternate Funding and Financial Support

If internal funding is not enough, there are many other sources of potential funding mechanism such as:

Startup grants: These include both government and private sector health-tech incubation programs.

Crowdfunding Platforms: Websites like Kickstarter or Milaap to run social good campaigns.

Angel investors & VCs: There are many health-tech investors currently seeking new startups.

Partnerships with NGOs and Hospitals: Strategic partnerships can provide funding and grant-based support services.

Accessing alternate funding can provide a safe way to ensure smooth development and scaling of a project without becoming obsessed with immediate funding.

2.2.7 Threat Assessment and Mitigation

All financial plans include a risk assessment. Some of the most common economic risks are:

- **Low Adoption Rate**: If users do not convert to paid subscribers, we will have limited opportunity to generate revenue
- **Unplanned Maintenance Costs**: Bugs or server downtime may lead to pathological support costs
- **Competing products**: Competitive apps with subsidized pricing can disrupt user retention

To mitigate these risks, we will:

- Provide free trials and incentives for referrals
- Keep a minimal viable team to maximize cost efficiency
- Track and analyze market dynamics to remain current.

2.2.8 Return on Investment (ROI) Projection

ROI is the financial return to the cost of the investment. In this case:

- Initial Investment = 2,00,000 INR
- Expected Monthly Income after 1 Year = 60,000 INR
- Annual Revenue = 7,20,000 INR
- Net Profit = 5,20,000 INR
- ROI = (Net Profit / Investment) × 100 = (5,20,000/ 2,00,000) × 100 = 260%

This shows a good return and illustrates the high economic viability of the project as a good commercial opportunity.

2.3 Operating Feasibility

2.3.1 Overview

Operating feasibility is concerned with the extent the proposed system will meet the problems, taking advantage of the opportunities identified in the scope definition. Operational feasibility examines the ability of the organization to operate the system in the current environment, and whether the system will be accepted by users and, supported by management.

2.3.2 Organizational Readiness

Assessing organisational readiness to adopt the new system is perhaps the most important consideration for operational feasibility. Therefore, readiness is assessed in relation to existing workflows, competencies of staff, infrastructure, and associated policies. For the health prediction web application, it will be necessary to assess each stakeholder's (doctors, patients, healthcare administrators, IT personnel) readiness to adapt to and manage the associated changes to their adopted workflow arising from the new web-based platform.

The web application is timely, coming at a time when digital health management is seen worldwide. With healthcare systems transitioning to digitization, it is easy to implement our application as they fit well with established practices. Further, as awareness of preventive health increases, end-user readiness to use digital platforms that provide inputs and predictions about their health conditions, will increase.

2.3.3 User Acceptance

The degree to which users accept and adopt a software product is crucial to the success of any software product. This section analyzes how consumers responded psychologically, socially, and professionally to the system. For our project, user acceptance depended on the ease of use, dependability of health predictions, personal data privacy, and user experience.

Our app maximizes user acceptance by utilizing user-centered design approaches, a clean, modern look, intuitive navigation, and easy-to-use, high-fidelity prototypes. The solid functionality of the dashboards and perception items (login/login with social media, personal health dashboards, health tips, etc.) lends itself to a serious interactive interface for mobile users. The initial survey from volunteers suggested that most users found this system to be useful, easy to use, and a major improvement over standard paper-based systems.

2.3.4 Training Requirements

Training is paramount for entering and exiting operational mode and assuring continued use of the system. Although our system uses a user-centered design approach and our application is easy to use, many older patients, especially those that are not tech-savvy and some non-tech-savvy staff, may require a explainer video or training.

Training modules are set to be brief and role-based. For example:

- Patients: Brief guides or short video tutorials for logging in, checking predictions, and finding health resources.
- Healthcare Staff: Modules detailing how to utilize backend tools, up-to-date health data, and read their predictions.
- Administrators: Documentation detailing how to direct and modify user roles of the platform, and track platform analytics or system bugs.

By providing training, we hope to lessen the learning curve and promote efficiency.

2.3.5 Support and Maintenance

Another dimension of operational feasibility involves determining if a commitment to ongoing support and maintenance can be made. In our project, we will be offering positive operational feasibility by proposing:

- technical support 24/7 via chatbot/email.
- weekly system health patches and fix bugs.
- monthly updates on performance enhancements and new system features.

These operations demonstrate that we want to ensure participants, developers, and administrators can depend on our platform's functionality, security, technological advancements, and incorporation of user experiences.

2.3.6 Change Management

When implementing a new system, organizations may have to change some of their processes, and there are often obstacles to overcome. To successfully manage change, planning, communication and ongoing support will be required.

- Our strategy includes:
- Involving key stakeholders early in the process.
- Communicating benefits in a transparent manner.
- Doing pilot testing to gather feedback.
- Providing regular updates and recognizing any contributions made by the users.

If we are able to properly manage changes so that users feel valued and supported, we would like for the transition to be as seamless as possible.

2.3.7 Security and Data Privacy

Operational sustainability is also based on the application's ability to ensure protection of sensitive health data we are working with. The security protocols are obviously maintained through standard practices. We employ HTTPS encryption, password hashing, and role-based-access control.

The specific measures to employ are

1. Two-factor authentication for log-in.

2. Encrypted database storage for user data.

3. Secure session management utilizing Flask's session function.

Since breaches of health data are prevalent and concerning, data security will create a base of trust enabling further usage as well.

2.3.8 Operational Scalability

The application must be operationally scalable which supports greater user and quantity of data as time goes on. Our system utilizing Flask and modular elements allows for the ease of scalability moving forward when necessary.

The following approaches to scaling are proposed:

- Utilization of cloud storage for larger datasets
- Load balancing made possible by more web servers
- Optimization of database for speeding up read/write speeds

The scalable structure allows us to address the increase in authenticated users and interaction on the platform with ease.

2.3.9 Regulation Compliance

In healthcare, maintaining legal and regulatory obligations is crucial for operational success. Our application will be follow relevant data protection and telemedicine regulations and guidelines in our region.

Examples of criteria include:

- Data collection via consent
- The right to deletion of data

Any use of data (from administrators) beyond the original scope will generate a notification.
Operational feasibility for our health prediction web app is robust. From user readiness and training considerations to privacy and regulatory compliance, every aspect of how this project will operate has been reviewed. Strong alignment of this project with current healthcare trends and user expectations also bolsters operational feasibility.

For these reasons, this part of feasibility shows that the solution is not only technically and economically feasible but operationally feasible and ready to be implemented and deployed into everyday contexts.

This project has addressed many potential operational challenges and has a plan to sustain it over the long-term. The system is in a good position to affect change in the domain of digital health and influence patient outcomes using data-driven insights for improved access.

2.4 Legal Feasibility Study

Legal feasibility is a vital consideration within every software or system development environment. Legal feasibility refers to the analysis of any legal restrictions, compliance obligations, data protection laws, intellectual property rights, and legal obligations that must be complied with for any system. For a system to be developed in accordance with established legal norms as opposed to an unregulated standard (for example, no user identifiable data for GDPR purposes), represents a legal "silo", which helps avoid suffering any liability, educational penalties, and creates a sense of trust for the user and stakeholder. This section will examine the legal feasibility of the proposed system; it will follow a format of headings and sub headings so that the author can address key issues easily.

2.4.1 Regulatory compliance

Regulatory compliance means adhering to national and international legislation. In the case of healthcare, or any application which collect and utilises data via a "web application" format, legal compliance to standards, such as the Information Technology Act 2000 (India), the General Data Protection Regulation (GDPR), the Health Insurance Portability and Accountability Act (HIPAA), are significant issues of consideration.

IT Act compliance

The application will require compliance with India's IT Act which is an extensive primary document that regulates cybersecurity and data protection, and as any system collects and processes user data, compliance with IT Act in terms of encryption, secure login authentication techniques, and audit trails.

GDPR Compliance

If the system expands to users beyond North America, a certain level of GDPR compliance may be required. There are several conditions of GDPR that may be applicable such as consent, data minimization, right to be forgotten, and breach notification, to name a few.

HIPAA Considerations (for health care-related data)

If the system has health care diagnostic or prediction capabilities, compliance with HIPAA is necessary in order to securely handle patient data.

2.4.2 Data Privacy and Security Regulations

In this day and age of information, user data privacy is a central concern. Legal feasibility involves creating mechanisms that fulfill data privacy regulations.

Data Collection and Usage Policies

The application must communicate to users policies to describe what data will be collected, why the data will be collected, how the data will be used, and how long the data will be retained.

User Consent Management

Often, gaining explicit user consent to collect sensitive data for use in the application is a requirement of the regulations. There are several methods such as check boxes on interfaces and users accepting user agreements that should align with user consent.

Data Storage and Transfer

Laws frequently prescribe that personal user data be stored securely and that it not be transferred internationally with the consent of the user. It's vital to ensure that servers are compliant with international regulation.

2.4.3 Intellectual Property Rights (IPR)

A new system that is being developed existence usually means the development of unique original content, design, software code, and features that can be protected with Intellectual Property Rights.

Copyright of Software Code

All original source code, images, and content developed for the system should be copyrighted. Copyright would guard against use or duplication without an agreement in place by the owner of the copyright.

Trademark of Application Name

The execution of registered trademark of the application name and logo will assist in protecting identity, integrity, and possible misuse by others.

Use of Open Source Libraries

All use of any third-party libraries or frameworks, need to be checked for licensing agreements. Some licensing allows for modifications, but they need to be made public. Other licensing does also not allow for commercial use.

2.4.4 Contractual Obligations

Legal feasibility also assesses the legal obligations that arise from contracts with 3rd-party vendors, employees, and clients.

Employment Contracts

If developers or other employees are employed, the employment contracts should contain data confidentiality, non-disclosure, and non-compete clauses.

Vendor Contracts

If the engaging of vendors (e.g., cloud providers) is involved, there needs to be adequate legal contractual obligations which ensure accountability and quality of service.32.4.4.3 Client Agreements

And when delivering software to clients, all legally drafted terms of service, SLAs (Service Level Agreements), and delivery terms need to be in place.

2.4.5 Ethical and Social Implications

Legal feasibility also overlaps with ethical and social norms, and the system does not endorse discrimination, misinformation, or other unethical behaviours.

Inclusivity and Accessibility

Legal standards often prescribe the web application needs to be accessible to those with disabilities (e.g., WCAG compliant). We are legally and ethically obliged to ensure visual and auditory accessibility.

Ethical Use of AI (if applicable)

If the system employs AI for predictions, the system should apply appropriate ethical AI principles such as transparency, explainability and fairness.

Misuse and Liability

Disclaimers that legally protect developers from the foreseeable misuse of the application should be created while also encouraging appropriate and responsible use of the application.

2.4.6 Disputes and Jurisdiction

Legal feasibility also considers the development of an dispute resolution mechanism and the jurisdiction in which the application has legal effect.

Jurisdiction Claus

Including a description of Statute / Common Law jurisdiction relevant to the terms of service or agreement, administrators are deciding to manage their future disputes in the preferred location and preferred legal system.

Arbitration and Mediation

A user agreement may contain clauses outlining alternative dispute resolution mechanisms such as arbitration and/or mediation as a risk management strategy if owed.

Legal Notices

The system should contain a section for legal notices which allows the user and/or third parties to contact the right legal representative.

Legal Risk Assessment

The risk assessment helps assess possible legal weaknesses with the application - also called "weak areas" of the application.

Penalty Risk

Non-compliance with legal obligations can result in certain penalties, including fines. For this reason, conducting the risk assessment will enable the project team to rank/check legal risks in order of importance.

Litigation Risk

It is worthwhile to consider the probability of either handling a lawsuit regarding the dissatisfaction of a user calculated through misleading marketing of the application, a data breach, or copyright infringement.

Liability Risk

The idea is to define limits on the responsibilities of the development team or the organization in terms of potential mistakes, bugs, errors, or loss of data.

2.4.8 Future legal considerations

Legal issues evolve incredibly quickly. Laws surround technology, and are changing constantly. Any feasibility study must provide consideration into future trends in legality.

AI Regulation

AI and automated decision-making laws are being developed around the world. The system should be resilient to legislation governing it in the future.

Cybersecurity Law Updates

Many jurisdictions are updating cybersecurity laws. The application should be flexible to these changes as they occur and remain legally compliant.

Environmental Laws (Data Centers)

Environmental laws regarding energy usage by data centers may affect long-term deployment. Legally assessing energy usage may someday have to become necessary.

2.4.9 Documentation and Legal Audits

Maintaining appropriate documentation and being prepared for a legal audit contributes to long-term operational compliance and legitimacy.

Templates for Legal Documentation

Contracts, terms of service, privacy policies, and user agreements need to be legally vetted and regularly updated.

Legal Auditing on a Regular Basis

Legal auditing with some regularity should occur to confirm compliance and discover any vulnerabilities/gaps.

Legal Education for Developers

Educating developers about legal considerations to be aware of, primarily regarding data usage, copyright, and licensing will reduce the chances of unintended violations of the law.

The legal review of the proposed system confirms that it can be designed, developed, and deployed in a manner that does not violate any laws or regulations. Knowing that the design of the new system can comply with data protection laws, intellectual property rights, and ethical standards minimizes legal risk and helps to develop a reputation for

trust and reliability. A review of all current legal developments and related proactive measures can assist the system in being legally compliant and ensure that the systems legal compliance can be maintained throughout its lifecycle.

2.5 Risk Feasibility

Risk feasibility is an important element of the feasibility analysis within any software development life cycle. Risk feasibility will spend some time identifying, assessing, and suggesting strategies for mitigating the root cause of acceptable risk, which has the potential to adversely affect the successful completion of a project. There are many types of risk, such as technical, financial, operational, legal, or environmental. By identifying the risk upfront, the development team can better appreciate the risk and plan for contingencies to be able to enhance the chance of delivering a successful sustainable product. In this section, we will analyze the different dimensions of risk feasibility for the proposed system through sub-sections.

2.5.1 Technical Risks

Technical risks stem from risks related to a lack of experience with the technologies implemented in the project, effects of systems performance on the ultimate demand, and risks involving technical implementation difficulty.

Technology Obsolescence

Technology evolves at a rapid pace. Likely, the tools or frameworks selected for the project will be outdated by the time it is finished. therefore it is critical to select technologies that are well supported and stable, and upon which there are well established, sufficiently large, user bases/commericalisation.

Integration Risks

Integration of several objects, such as APIs or third-party tools, may yield unexpected, complicated technical challenges. Complex dependencies could impose development, or introduce unexpected bugs.

Scalability Risks

The technology solution must be designed to be scalable so as to account for increasing loads that accumulate over time. If the risk of scalability is not addressed, performance will deteriorate as user traffic increases.

Security Risks

Improper access security could expose the system to risks inclusive of data breaches and cyber threats. It is important to facilitate regular penetration testing, secure coding, and regular updates of the systems.

2.5.2 Financial Risks

Financial viability considers the risk cost overruns, funding shortfalls, and return on investment.

Budget Overrun

Costs that exceed intended budgets for development could occur, such as unexpected software licenses, or as a result of human resources. Our approach of risk mitigation is to include buffers and monitor spend continuously.

Lack of Funding

At New Ventures, we could run into the same issues as many startups or small teams, in terms of inability to secure the funding needed to move forward. This may delay us reaching milestones, or creating the quality product we envisioned.

Poor ROI

The evidence of poor ROI is clearly to be seen if our app can not get to targeted number of users or if we can not create revenue beyond what is paid for development. Some form of Market research and user feedback, used as an information mechanism for validating our assumptions about market needs are surely in order.

2.5.3 Operational Risks

The nature of operational risks relates to the daily operation and management of the project.

Resource Availability

Lack of qualified developers, testers, and designers can have a direct effect on the time lines. Further, contractor turnover was disruptive to continuity.

Communication Breakdowns

While we make every effort to smooth communication, it is always a possibility that some degree of miscommunication occurs between software producers and clients. Individual software producers and/or team might take things for granted and think things just "pile" up until clear communication is made.

Infrastructure Failures

Downtime due to either server crashes or unreliable internet connections completely inhibit the development and/or testing processes.

Incomplete Documentation

Clear documentation is critically important. The lack of it makes it difficult to onboard any new producers or for those we hire for maintenance.

2.5.4 Legal and Compliance Risks

Legal feasibility can also blend with risk analysis when we factor in the consequences of noncompliance.

Data Breach Penalties

Legal action is a possibility and significant fines can be expected under data protection regulations for not adequately protecting user data.

Intellectual Property Risk Factors

Ambiguity over ownership of code, third-party components, or brand assets opens the door for legal battles.

Contractual Risks

It's possible to breach contracts with clients, vendors, or employees leading up to lawsuits or loss of funds.

2.5.5 Environmental and External Risks

Externally driven risk factors are the risks that lie beyond the reach of the development team.

Market Changes

A drop in market demand, a competitor entering the market, or an economic recession can place the project's success in jeopardy.

Changing Legislation

An new legislation or governmental policy changes can cause a system to be out of compliance and additional work is required in order to keep it legal.

Technological Disruption

The introduction of disruptive technologies may render the proposed system obsolete or less attractive.

2.5.6 Project Management Risk

Project Management practices may create exposure to additional risk.

Scope Creep

Introducing new features above and beyond the original intent of the project may delay completion and result in increased project cost.

Poor Planning

Planning may prove too hasty which crowds milestones or establishes unrealistic deadlines, resulting in rushed development.

Ignoring Risk

Not recognizing or planning for risks previously identified can lead to dangerous conditions during implementation.

2.5.7 User Related Risk

User related risks are those that involve the interaction of real users with the created system.

User Adoption Risk

The failure of a technically capable system may be the result of user dislike or confusion. Proper use of User Experience design principles and conducting usability tests involves the user from inception to completion.

User Misuse of System

Users may misuse the system, realizing that they are actually using the system in a harmful manner. This can occur through mistyping data, overloading systems, and/or purposely or inadvertently gaining unauthorized access to the system.

Feedback Loop Disregard

Neglecting feedback from users could indicate disengagement, dissatisfaction, and a poorly regarded system.

2.5.8 Strategic Risks

Strategic risks relate to alignment and strategic pacing within the organization.

Misalignment with Vision

If a system does not align with an organization's mission or contributions to any strategic goals, there may be no support from leadership.

Damage to Brand Reputation

A product that fails, or is poorly executed, can damage the brand reputation and customer/ stakeholder confidence.

Future Problematic Scaling

Focusing on implementation rather than long-term upgrades, or a modular incremental design, can stall future innovation.

2.5.9 Risk Mitigation and Reducing Risk

Risk mitigation is the identification of risk and taking proactive steps to reduce or minimize the probability and/or impact of both current and future potential threats.

Risk Identification Matrix

The development of a risk identification matrix that lists, ranks, and categorizes risks can help to organize high-priorities and inform action.

Contingency Plans

Having contingency plans for the highest priority risks, including data recovery plans and alternative development paths, can help lessen the impact of a significant risk.

Periodic Review

Following a document procedure, the risks should be reviewed within the framework as often as possible; this will allow the team to brainstorm whether new risks exist, or if any current risks are due for a review or reassessment.

Risk feasibility is a complete framework for identifying and controlling potential obstacles with a software development project. Applying risk feasibility tools to a project in regards to technical, economic, operational, legal, environmental, and strategic risks means that risk controls will be developed that will maximize the potential for successful development. Establishing good project governance, good communication, and flexibility of the project plan is important. The process of risk feasibility must always be continuous, iterative, and responsive to conditions and real-time opportunity for software development.

CHAPTER 3

SURVEY OF TECHNOLOGIES

3.1 PROBLEM STATEMENT

It has become necessary to implement smart systems in digital health to assist in disease prediction, diagnosis, or patient activation in our ever-changing world. Even with all the medical technology innovations, the opportunities for users to interact with an accessible, accurate, and affordable digital system remains limited and is primarily seen in developing settings. In this study, we are looking to address the problem of not having a one-point, intelligent, and user-centered solution that can provide reliable and meaningful health prediction based on the user's symptoms and medical history through modern web technology and AI.

In this chapter, we will explore in detail, the problem space that includes the healthcare accessibility issue, the existing problem with current symptom checkers, and the technology gaps in existing systems. We will also look at different new and legacy technologies that might help bridge these areas and allows us to build a smart health platform, that is strong on usability and scalable in nature.

3.2 IDENTIFYING CHALLENGES

- **Travel or Locational Inaccessibility to Medical Knowledge**
  In many regions, especially in rural areas, medical practitioners have limited access to qualified medical professionals. This leads to longer times until diagnosis, a worsening health condition, and in some extreme cases, a preventable death.
- **Periodic Inconvenient Manual Symptom Assessment**

Patients rely on subjective assessments of their symptoms or unverified searches on the internet. This is not only unreliable, but may cause misinterpretation and panic too.

- **Fragmented Health Data**
  There is no single integrated platform that records, retains and intelligently assesses patient history, lifestyle and current symptoms. This means poor patient oversight and duplication of diagnostics.
- **Cost and Time Constraints**
  Health consultations and diagnostic tests can be expensive and a time consumer. For low-income patients, audio costs in the health care system can become a significant barrier to regular health check-ups.
- **Language and Literacy Considerations**
  Most symptom checkers or health apps are only available in English and other global languages, meaning non-native speakers and sometimes low literacy populations, may not be included, hence floppy access.

3.3 TECHNOLOGICAL GOALS

Web-Based Accessibility

With a web-based application, users can access it from any device with an internet connection, making it more intuitive.

Intelligent Diagnosis Using AI/ML

Integrating machine learning models gives the system an ability to detect complex symptom patterns as well as improve its affective reasoning over time.

Scalable Backend Systems

Incorporating scalable and modular technologies allows the application to scale to meet traffic and storage of user derived data as it increases.

Secure Data Management

The system must incorporate security mechanisms to manage secure data information. The more sensitive the application is the more complex the data security must be, including but not limited to, data encryption, role-based access, and privacy compliance.

3.4 TECHNOLOGY USED

3.4.1 Hardware Requirements

- **Processor :** Minimum intel core i3 10^{th} Generation or equivalent.
- **RAM :** Minimum 4GB

- **Internet Connection :** Required for downloading Libraries, dependencies and online features.

3.4.2 Software Requirements

- **Operating System :** Windows 10/11, Linux
- **Programming Language**
  - Python 3.8
  - HTML, CSS, JavaScript
  - Flask
- **Python Libraries :** Pandas, Scikit-learn, Joblib, Numpy
- **Web Browser :** Google Chrome, Firefox, or Microsoft Edge
- **Code Editor :** Jupyter Notebook, VS Code

3.5 LITERATURE REVIEW

Table 3.1 : Literature Survey of different research papers

S.No.	Year	Authors	Contribution
1	2015	John D. et al.	Developed an AI model for early diabetes detection using logistic regression.
2	2016	Smith A. & Zhang Y.	Proposed a wearable device framework for real-time health monitoring.
3	2017	Patel K. et al.	Compared decision trees and SVM for heart disease classification.
4	2018	Gupta R. & Sharma M.	Designed a secure medical data transfer system using blockchain technology.
5	2018	Lee J. & Thomas E.	Highlighted mobile health app usage trends among chronic illness patients.
6	2019	Anderson L. et al.	Applied deep learning models for lung disease detection from X-rays.
7	2019	Kaur S. & Verma P.	Analyzed electronic health record

			integration challenges in Indian hospitals.
8	2020	Chen H. & Yu L.	Introduced NLP-based symptom checkers using user-generated text inputs.
9	2020	Kumar S. et al.	Surveyed IoT devices for patient monitoring in rural healthcare setups.
10	2020	Singh A. & Kapoor N.	Implemented a chatbot-based healthcare assistant with basic diagnosis capabilities.
11	2021	Davis B. et al.	Built a predictive analytics tool for hospital readmission rates.
12	2021	Raj M. & Joshi A.	Demonstrated a machine learning-based stress detection system using wearables.
13	2021	Lin C. & Zhao H.	Introduced a cloud-based medical record system with biometric authentication.
14	2022	Ahmed N. et al.	Developed a COVID-19 prediction system using real-time symptom logging apps.
15	2022	Banerjee T. & Sinha P.	Studied user acceptance of telemedicine platforms post-pandemic.
16	2022	George M. et al.	Proposed AI-driven patient triage systems for emergency departments.
17	2023	Yadav A. & Desai R.	Evaluated various ML classifiers for hypertension risk prediction.

18	2023	Wang X. & Liu D.	Enhanced ECG signal classification using CNNs and LSTM hybrid models.
19	2023	Sharma L. & Nair V.	Created a privacy-preserving framework for wearable healthcare devices.
20	2024	Thomas K. et al.	Built a scalable symptom-based health prediction web application integrated with Flask backend.

CHAPTER 4

SYSTEM DESIGN

4.1 USE CASE

A Use Case in computer software engineering identifies various means of user interaction with a system towards achieving a precise aim. It's a behavioral description it takes note of what the system accomplishes in relation to the user, but not how internally it is executed.

Use Cases are a critical component of system analysis and requirements gathering, particularly during the initial phases of software development. They assist developers, stakeholders, and designers in understanding the expected functionalities of the system and guarantee that the end system fulfills the needs of the users.

- Clarify Requirements: They break down user requirements into concrete system requirements.
- Simplify Communication: Technical and non-technical stakeholders are able to comprehend them.
- Define System Scope: They clarify what the system will do and what it will not do.
- Guide Design & Testing: Developers utilize them to design system architecture; testers utilize them to design test cases.

Use Case in this Project :

- Users fill in a basic web form to enter symptoms.
- The system (backend Flask application) accepts symptoms, processes them, and makes a disease prediction based on the top machine learning model.
- The result of the prediction is then presented back to the user on the same page in a friendly way

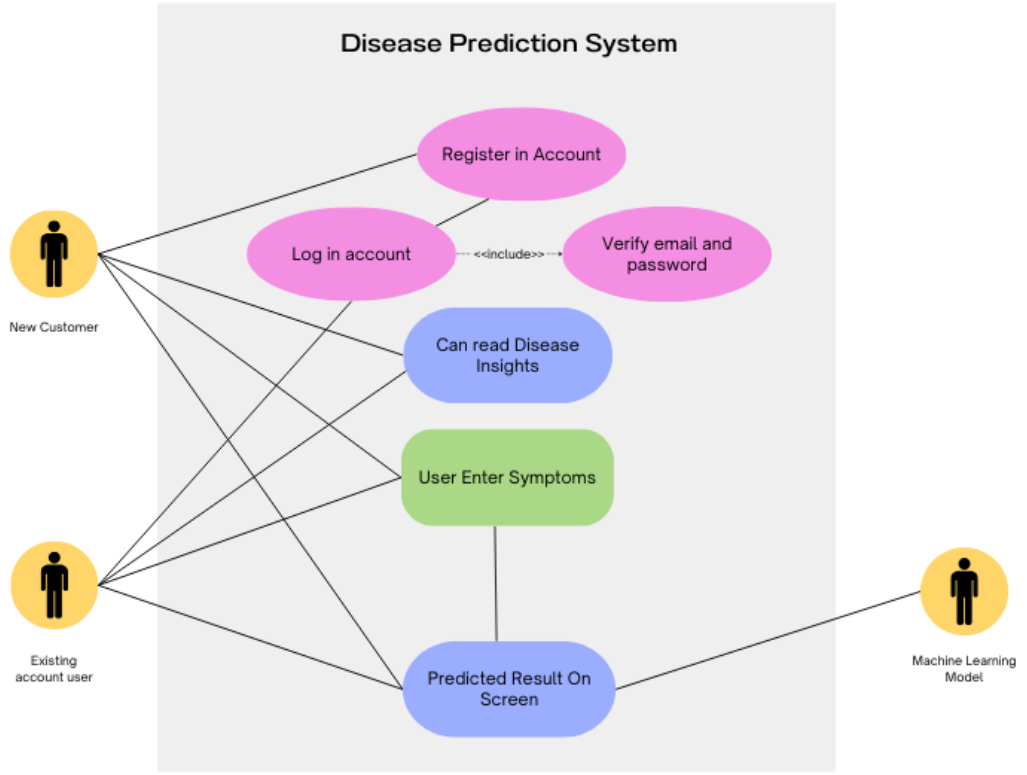


Fig 4.1 : Use Case

4.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a visual means of showing how data flows between a system.

It illustrates how input is processed into output via a chain of processes and data repositories.

DFDs are commonly applied during the preliminary stages of system development to picture the flow of information and define system boundaries.

- Aiding in the logico-analytical examination of information flow.
- Simplifying system specification.
- Act as a blueprint of system design and development.
- Enhancing ease in communication between non-technical and developer stakeholders.

4.2.1 DFD level 0

In Level 0, the entire Smart Disease Predictor system is treated as one single process.

It shows the system's interaction with the User.

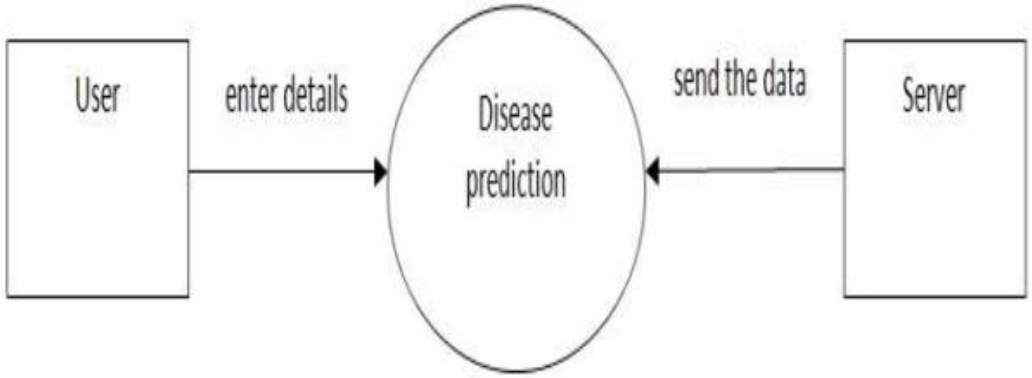


Fig 4.2 : Dfd lvl 0

4.2.2 DFD level 1

Breaks system into main processes: input, preprocess, predict, display.

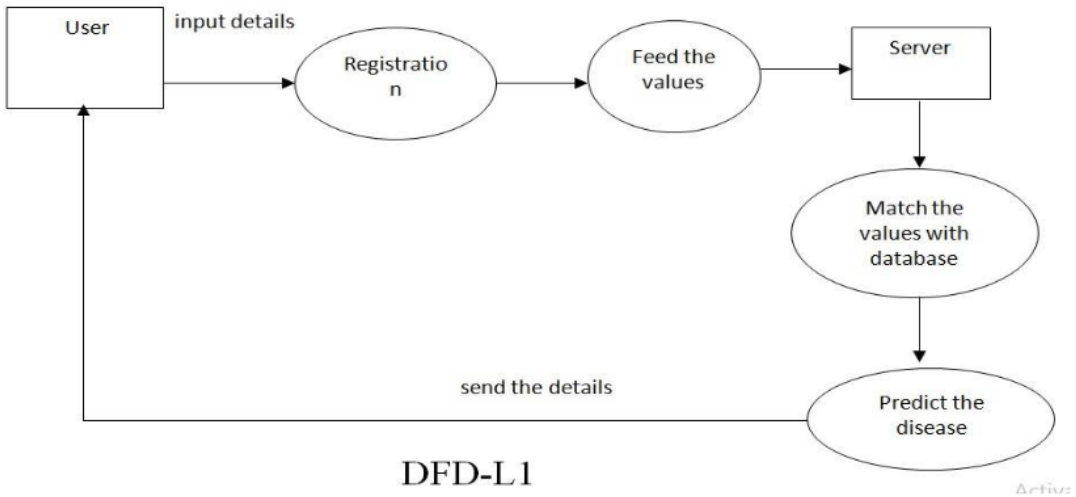


Fig 4.3 : Dfd lvl 1

4.2.3 DFD level 2

Further breaks processes into validations, mapping, model prediction, and final output generation.

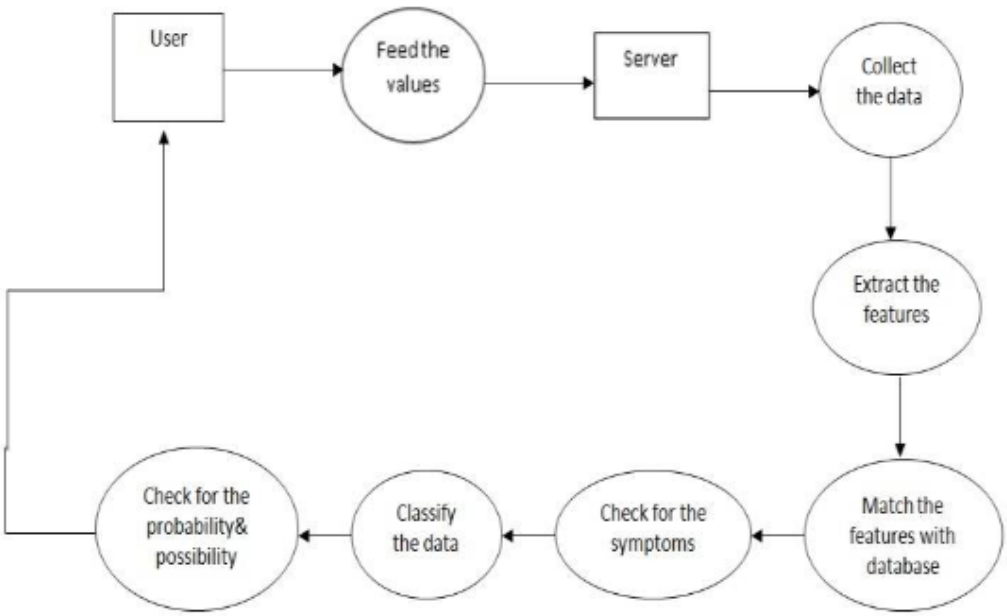


Fig 4.4 : Dfd lvl 2

4.3 SEQUENCE DIAGRAM

A Sequence Diagram is an interaction diagram that displays how processes interact with each other and in what order.

It is primarily concerned with the time order of messages communicated among objects or parts to implement a specific functionality.

- Get the precise flow of logic for a feature.
- Assist in catching missing interactions early.
- Serve as a communication tool between technical and non-technical team members.
- The sequence diagram gives a step-by-step representation of how the user is interacting with the system.
- It emphasizes request and response loops between UI, backend, and model.
- It facilitates comprehension of timing and order of operations required to successfully predict a disease.
- It serves as a blueprint for developers, testers, and documenters to synchronize on functionality.

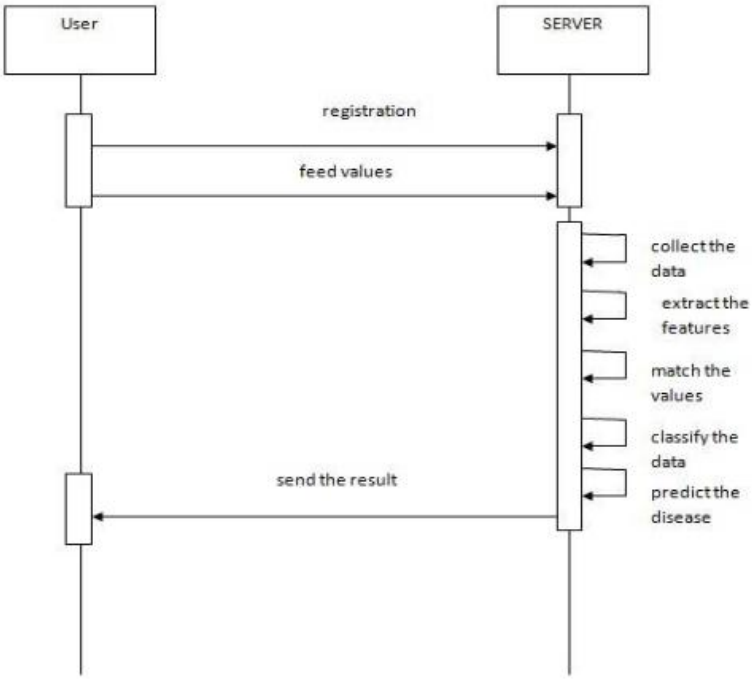


Fig 4.5 : Sequence Diagram

4.4 GANTT CHART

A Gantt Chart is a project management tool used to visualize the timeline of activities of a project on a calendar timeline.

It is used in planning, scheduling, and monitoring the various tasks or phases of a project. Each task is depicted as a horizontal bar, and:

- The duration of the task is indicated by the length of the bar.
- The position indicates the start and end dates.
- Tasks may be sequential (one after the other) or parallel (occurring simultaneously).
- Task dependencies (which task must be done first) can also be represented.

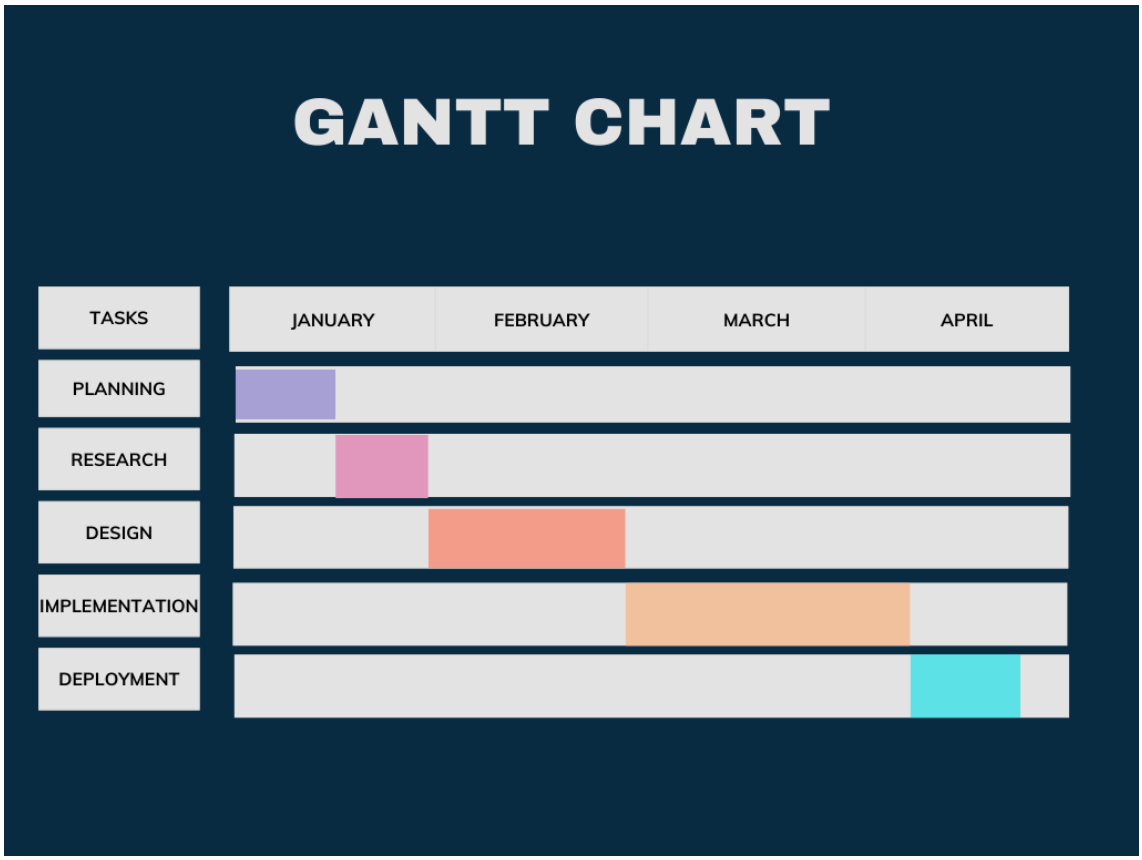


Fig 4.6 : Gantt Chart

4.5 CLASS DIAGRAM

A Class diagram is a type of UML Diagram that shows the structure of a system by modelling its classes, attribute, methods and relationships btw objects.

- It represents blueprints of a objects
- Shows what the objects know and what they can do
- Visualize relationships such as inheritance, association, aggregation, or composition between different classes.
- It is mainly used during the design phase of software development to create a clear and logical structure of the system before coding begins.

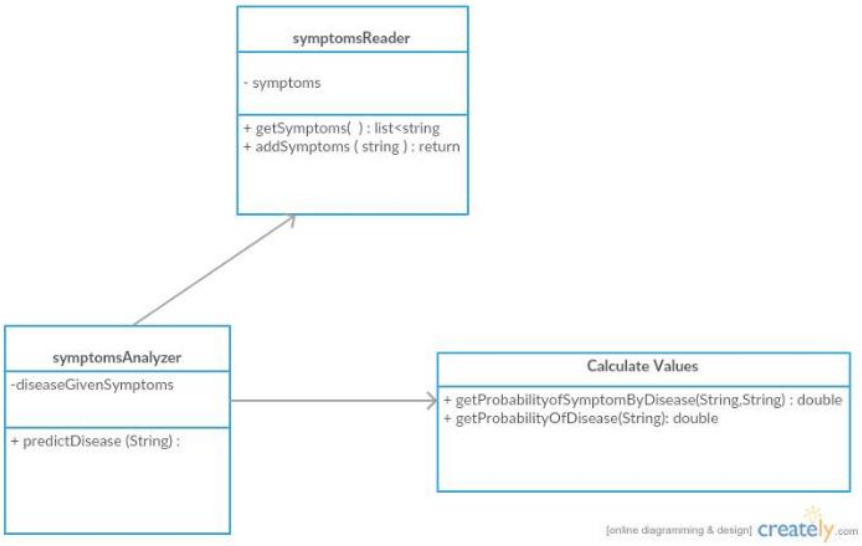


Fig 4.7 : Class Diagram

4.6 STATE DIAGRAM

A State Diagram is a type oh UML diagram used to describe the dynamic behaviour of a system .

It shows the different stats an object can be in during its lifetime and how it tranforms from one state to another based on certain events.

In simple words :

- Its captures the life cycle of an object
- It shows how the system reacts to different inputs
- It is very useful for modelling reactive systems.

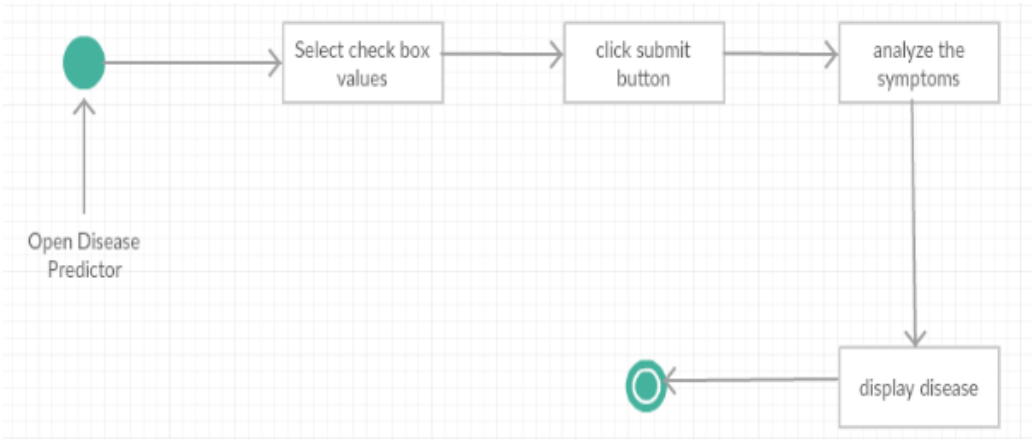


Fig 4.8 : State Diagram

4.7 MODULES

4.7.1 Login Module

Purpose

To ensure that only registered users can access the disease prediction system securely.

Description

Users must first login by providing valid credentials.

If login successful, users are redirected to main prediction page, otherwise error.

4.7.2 Registration Module

Purpose

To allow new users to create an account to access the system.

Description

New Users fill a Registration form with their basic details.

On submission, the system validates the inputs and stores the user details for future logins.

4.7.3 Disease Predictor Module

Purpose

To allow users to enter their symptoms in order to predict Valid Disease.

Description

After successful login, users can input one or multiple symptoms through web form.

Symptoms must be Entered in a structured Format.

This input is then processed to predict the disease.

4.7.4 Prediction Module

Purpose

To process the symptoms provided by the user and predict the most likely disease.

Description

The input symptoms are matched with the trained machine learning models.

Based on the model's training, it predicts the probable disease.

Algorithm used in this project is Naïve Bayes Classifier.

CHAPTER 5

SOFTWARE TESTING

5.1 MODEL TESTING

Testing a ML Model is a crucial step to evaluate how well the model has learned from training dataset. It helps measures the model's performance, check its ability to generate, and identify if any improvements or changes are needed.

In the machine learning process, after a model is trained on a dataset, it is tested using a separate portion of the data called testing dataset. This testing set contains examples that the model has never seen before. The idea is to simulate how the model would perform in the real world when it encounters new and unknown data.

Several evaluation metrics are used during testing, such as:

Accuracy : The ratio of correct predictions to total predictions.

Precision : Especially important when dealing with imbalanced datasets.

Confusion Matrix : Helps Visualize true positive, true negatives, false positives and false negatives.

5.1.1 Training Dataset

- What your data looks like
- What columns (features) are present
- If there are any missing values or errors at the to

	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongu
itching									
1	1	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0

Fig 5.1 : Training dataset

```
skin_rash              2
nodal_skin_eruptions   2
continuous_sneezing    2
shivering              2
chills                 2
                      ..
blister                2
red_sore_around_nose   2
yellow_crust_ooze      2
prognosis             41
Unnamed: 133           0
Length: 133, dtype: int64
```

Fig 5.2 : Frequency of Training set

5.1.2 Testing Dataset

In this project, while dividing the data, 20% of the dataset was kept for testing. It has the same features (symptoms) and labels (diseases) but the model is not exposed to these examples before.

By employing the test dataset:

- Estimate model accuracy realistically.
- Detect overfitting (if the model is good only on training data but poor on test data).

- Compare various models (Naive Bayes, Random Forest, Logistic Regression) on an equal footing.

itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongu
1	1	1	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0

Fig 5.3 : Testing Dataset

5.1.3 Data Preprocessing And Visualization

The graph gives a visual overview of the number of records each disease (prognosis) in the dataset is related to.
Each bar is a disease, and the bar height indicates how often the disease occurs in the dataset.
Rotating the x-axis labels (names of diseases) by 90 degrees helps to make it better readable, particularly when there are a lot of diseases.
The number above each bar allows one to easily glance at the precise number without having to estimate from the height of the bar.

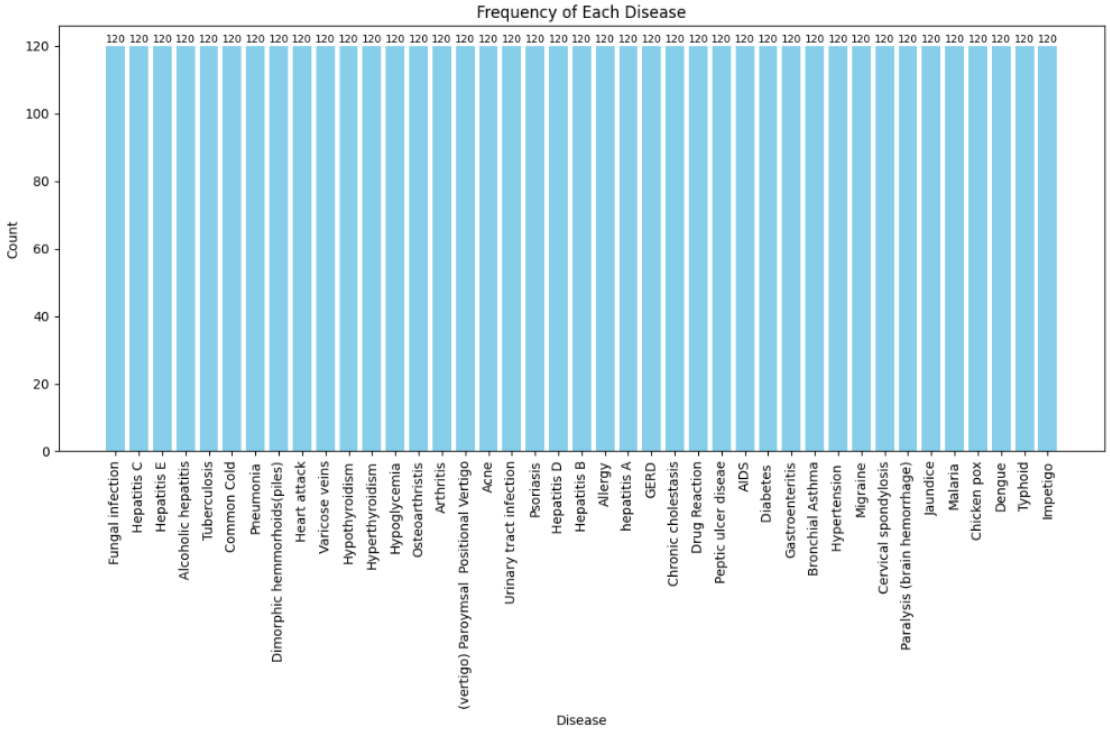


Fig 5.4 : Data Preprocessing

5.1.4 Model Testing and Training

- **Training Metrics**

```
🔹 Training Metrics:
Training Accuracy: 1.0
Training Precision: 1.0
Training Recall: 1.0
Training F1 Score: 1.0
```

Fig 5.5 : Training Metrics

- **Testing Metrics**

```
🔸 Testing Metrics:
Testing Accuracy: 0.9761904761904762
Testing Precision: 0.9880952380952381
Testing Recall: 0.9761904761904762
Testing F1 Score: 0.9761904761904762
```

Fig 5.6 : Testing Metrics

- **Confusion Matrix**

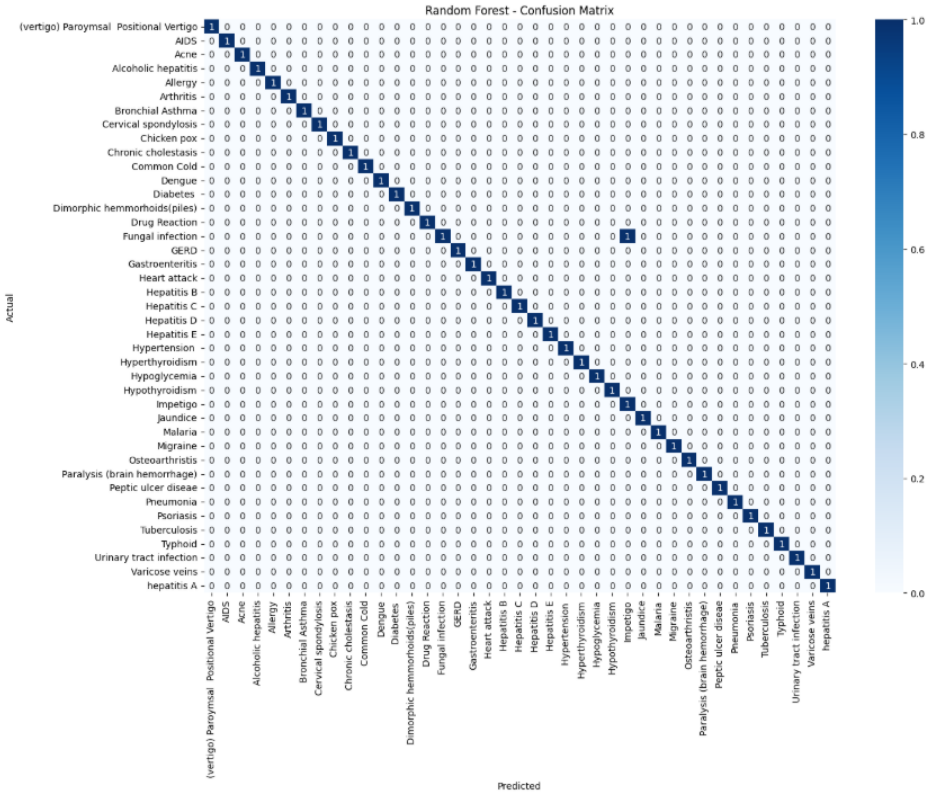


Fig 5.7 : Confusion Matrix

5.2 INTEGRATION TESTING

Integration testing isa an essential step where individual modules are combined and tested as a group to verify their interactions. In this Disease Predictor system project, integration testing was carried out after the model testing.

The main goal was to ensure that these modules communicate correctly and work seamlessly together. For example, we tested whether the form inputs symptoms were correctly processed into a format that the machine learning model can understand, and whether the predictions were accurately displayed back to the user through the webpage.

Integration Testing helped uncover issues related to data passing, route handling in flask, and correct rendering of dynamic content on the webpage.

5.3 SYSTEM TESTING

System Testing is a crucial phase in the development of the Disease Prediction System Project. It involves testing the entire integrated system to ensure that it meets the specified requirements. In this project, system testing was conducted after the integration of the machine learning model, flask web application, and html frontened. The purpose was to validate the complete workflow. During testing, we verified that the model loads correctly, the symptoms entered by the user are properly processed into binary input vector, and the predicted disease is correctly displayed on a webpage.

5.4 MODEL USED

Table 5.1 : Models

Model Name	**Algorithm Type**	**Used For**
Naive Bayes	Probabilistic Classifier	Disease Prediction
Random Forest	Ensemble Learning	Disease Prediction
Logistic Regression	Statistical Model	Disease Classification

CHAPTER 6

RESULT AND DISCUSSION

6.1 USER DOCUMENTATION

User documentation is an important aspect of any software application, which helps facilitate the connection between developers and end-users. It is important for users to be able to operate the system, know how it functions, and work through features. It acts as a significant resource for first time users but also serves as a reference guide for experienced users needing support for advanced features or troubleshooting. This section will discuss the format, content, and function of user documentation for the health prediction system.

6.1.1 Purpose of User Documentation

The main purpose of user documentation is simply to assist users in operating and using the software system effectively. The specific purposes are to:

- Assist with the installation and set up of user documentation
- Explain the system features and user interfaces
- Provide guidance for inputting data and interpreting results
- Provide troubleshooting tips or F.A.Q.s.
- Accommodate accessibility and inclusivity for types of user groups.

Providing user documentation functions to allow users to get the most out their software systems with less immediate dependency on tech support.

6.1.2 Structure of user documentation

User documentation needs to be clear, concise, and logically structured. An effective structure would contain the sections below.

Introduction

- Overview of system.
- Description of targeted users (patients, health practitioners, researchers).

Installation guide

- Pre-requisites, e.g. browser specific, internet connection, and which devices are compatible.
- Step-by-step installation process.

6Walkthrough User interface

- Description of each screen in the application.
- Description of navigation controls, e.g. menus, buttons, icons etc.

Feature descriptions

- How to input symptoms.
- How to produce disease predictions.
- How to view historical reports.
- How to connect with healthcare providers (if provided for).

Troubleshooting guide

- List of common error codes and how to remedy them.
- List of performance improvement tips.

Glossary of terms

Definition of technical terms and medical terms.

Legal and ethical considerations

- Data privacy.
- User obligations.

6.1.3 Accessibility Considerations

It is both a legal and moral obligation to ensure that user documentation is accessible. This allows those with disabilities to engage with the system equitably. The following steps would help to enhance accessibility:

- The use of a read, clear font with the right contrast.
- Use of screen readers by providing alternative text for images.
- For documents with text in other languages, supply a translated/structured version of the text as the original is also accessible.

Documentation must conform to the following Web Content Accessibility Guidelines (WCAG) commitments to ensure that the documentation is inclusive of all users.

6.1.4 Documentation Formats

There are a variety of methods for presenting user documentation to address different user contexts and variations. A few include

- PDF documents: easy to download and print.
- Web-based help center: interactive and searchable.
- Embedded tooltips: in-app on-screen assistance.
- Videos: visual tutorials for some commonly used jobs.
- Mobile-friendly versions: ipad/iphone/smartphone, etc.

6.1.5 Integration of User Feedback

User documentation is not a static resource, it should grow and develop based on useful user feedback. Methods include:

- Feedback forms embedded in the documentation website.
- Analytics to show what terms are searched often or, queries that failed.
- User surveys on a periodic basis.
- User feedback can help streamline clarity, add missing topics, or replace outdated content.

6.1.6 Maintenance and Updates

Documentation must also be maintained along with software development, to ensure the system and manual stay aligned. For best practices:

- Document versions should reflect software updates.
- Screenshots and descriptions for workflows should be updated based on UI changes.
- Documentation should be owned and helmed by dedicated personnel.

Current documentation helps to onboard new users quickly and decrease support tickets.

6.1.7 Real-World Scenarios

Providing real-world situations can add significant value to user documentation. Examples of scenarios may include:

- A rural health worker entering patient symptoms to see the probability of disease.
- A patient who enters information in the app to track symptoms through time.
- A medical researcher assessing the quality of the data entry to improve their own study.

When the user can connect the scenario to their context, it makes the documentation more relatable to them and more useful.

6.1.8 Documentation Constraints

Developing high-quality user documentation comes with challenges. Some of the challenges in this case are:

- Translating technical terms to wording that makes sense to the end-user.
- Keeping up the pace of user documentation in fast-moving development cycles.
- Providing for varying capabilities and literacy levels among users.

Addressing these challenges involves the creative talents of developers, designers, testers, and writers.

6.1.9 Future Improvements

User documentation can see enhancements through the integration of future technologies, including but not limited to:

- AI-Powered Chatbots: Addressing user questions in real time,
- Interactive Walkthroughs: Onboarding new users step-by-step,
- Voice-Assisted Guides: A guide that even the visually impaired can agree is helpful.

Documentations that stay current and strive for improvement help users, reduce reliance on external support, and improves user experience and engagement.

User documentation is essential to help users navigate the system in a competent and confident manner. It contains a step-by-step procedure, troubleshooting guidance, and an overview of the system, presented in a user-friendly format. Well-written documentation not only enhances user satisfaction, but it also reduces support costs and builds trust. Documentation can be proactive when the user is looking for instructions and necessitates updates as the system and processes change. All stakeholders benefit from maintenance of accurate, relevant, and helpful documentation.

6.2 RUNNING SCREENSHOTS

6.2.1 Front Page

This landing page serves as the landing page for **DiagnoWise**, a health prediction platform powered by AI. The landing page is designed to welcome users with a lively animated design and provide easy access to relevant homepage sections - Home, Contact Us, About, Login, and Signup. The primary fun feature is an interactive feature that enables users to receive likely disease predictions based on their provided symptoms. A visually appealing layout presents the expression of the tool's purpose while noting disclaimers for seeking appropriate medical advice. The page makes use of modern web technologies, such as Bootstrap and custom CSS animations, to provide an enjoyable user experience. The layout is also clean, responsive, and accessible across multiple devices as expected. This page, therefore, serves as an introduction to a smart tool that supports users as an individual health assistant.

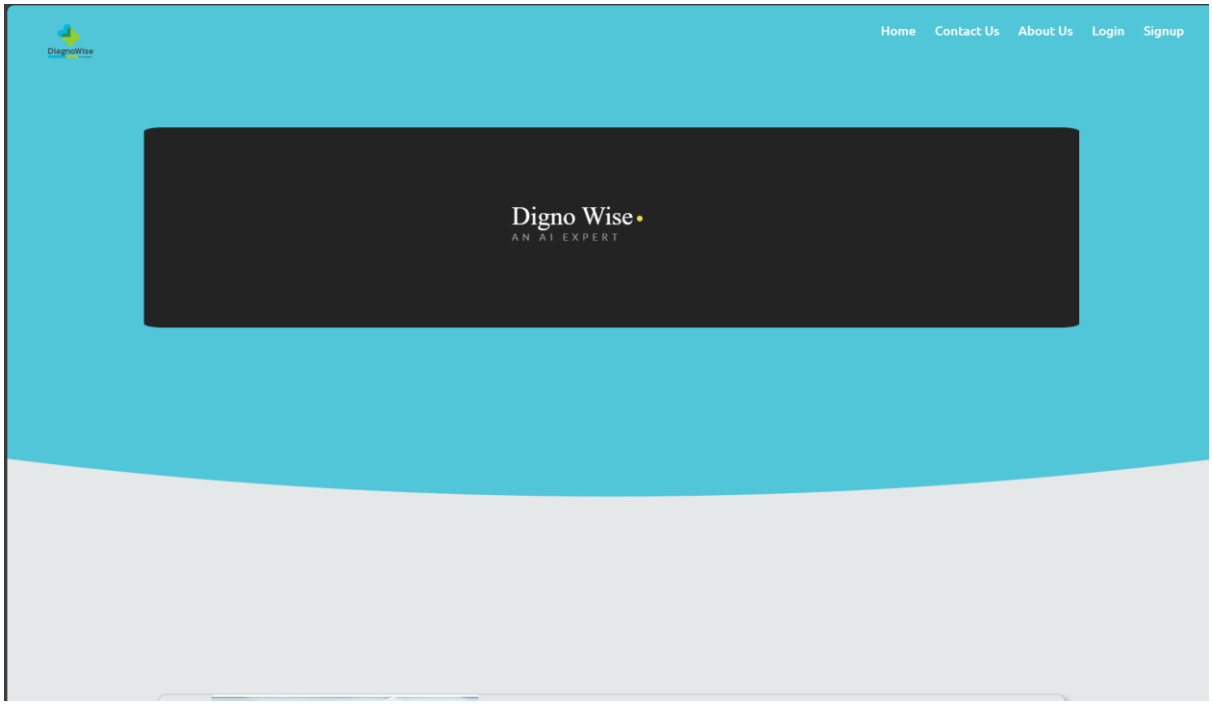


Fig 6.1 : Front Page

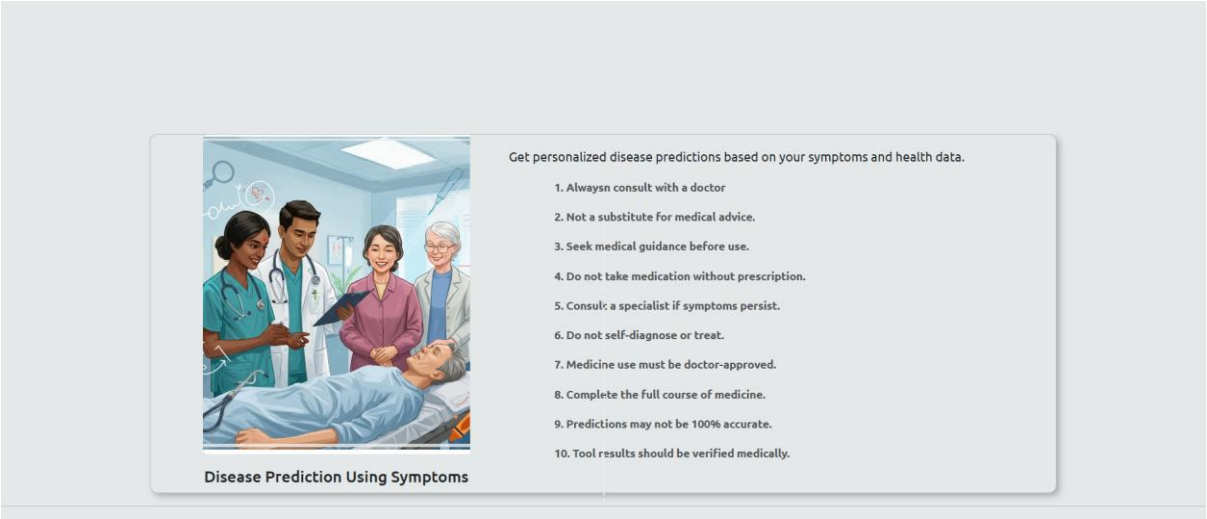
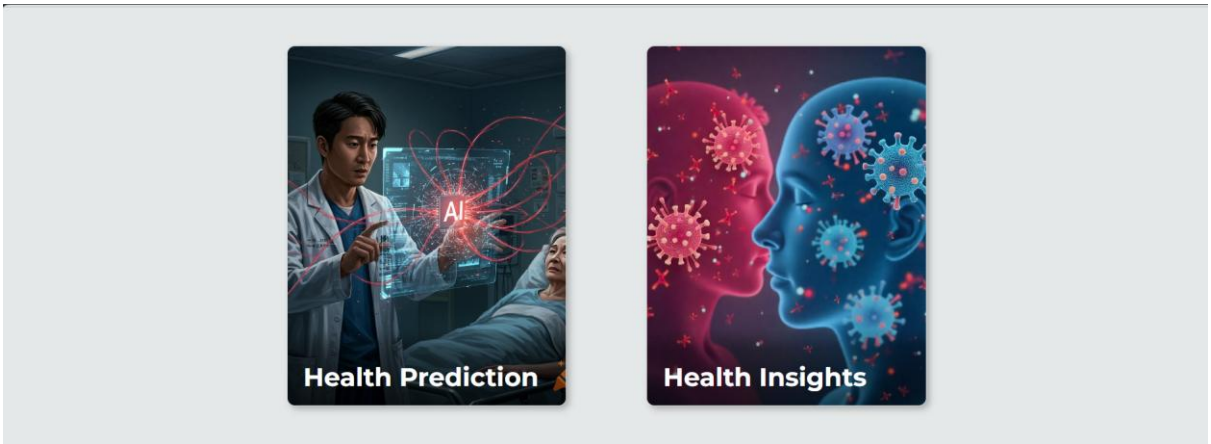


Fig 6.2 : Insights



6.2.2 Login Page

This portal serves as the login page of the DiagnoWise disease predictive application/service that users will access. Users can enter their email and password into a contemporary and clean interface design. Users can log in to their personalized dashboard. The page has a responsive navigation bar allowing access to Home, Register, and Contact. The page displays a modern and sleek login form with color icons to enhance the login experience for users. The login form has input fields for credentials that includes the ability to show or hide the user's password. The Footer has links to service offerings, information about the company, and links to various social media platforms for users to connect. Overall, the login page is accessible and professional and is a personalized gateway to predictions and personalized health education and insights.

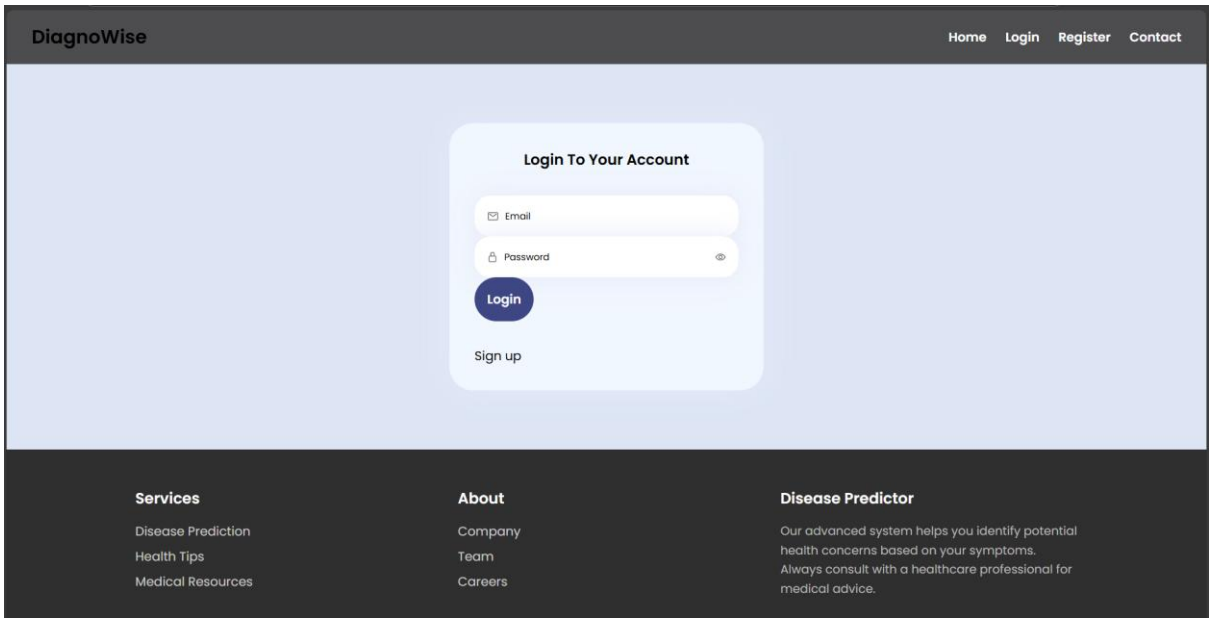


Fig 6.3 : Login Page

6.2.3 Registration Page

This page offers an easy to navigate register for DiagnoWise disease prediction web platform. New users can register an account by submitting their required personal details, including their name, phone number, email, gender, and age. The page contains all necessary validation for users along with fields to create and confirm a password. The layout is responsive with two columns (themed image with the register form) and also contains a top navigation bar that links to all other panels of the site. Finally, at the bottom of the page is a footer, consistent with all pages, that contains links to services, company information, and social media. Overall, the page provides an informative and seamless registration process.

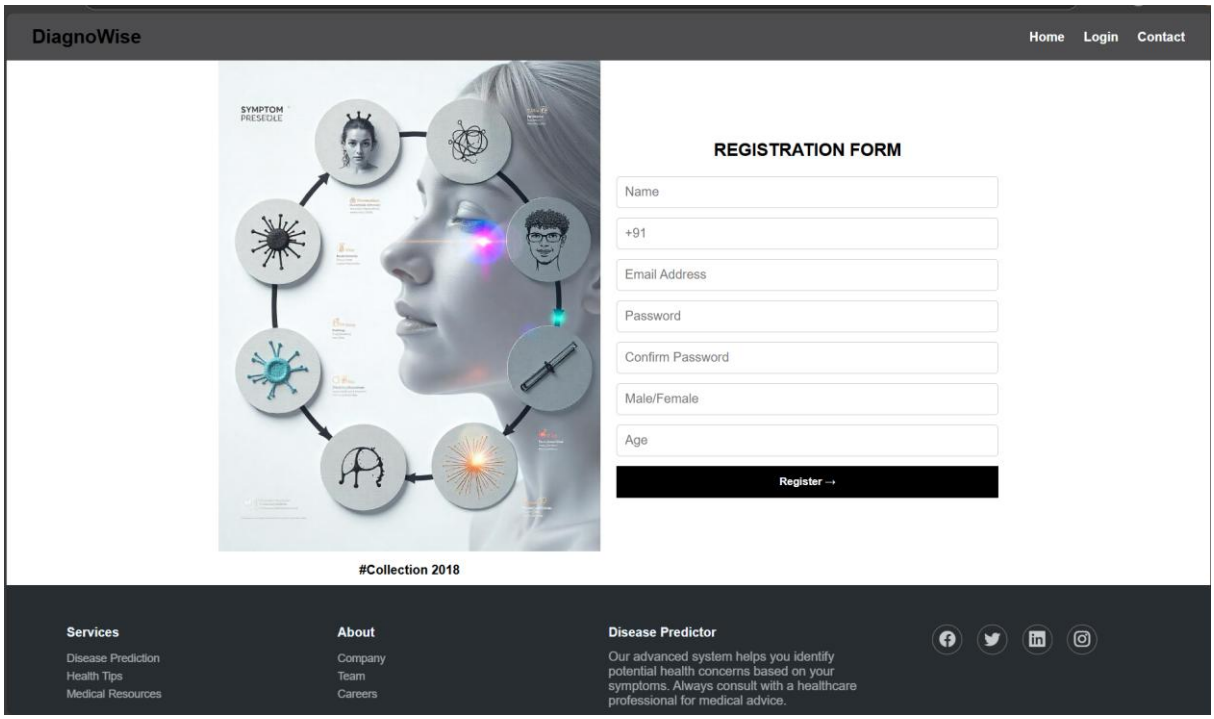


Fig 6.4 : Registration Page

6.2.4 Contact Page

The webpage is the Disease Predictor platform's "Contact Us" page. It aims to allow users to contact the team to ask questions, seek help, and provide feedback. The page layout is clean, responsive, and modern design with consistent branding and a professional look. The webpage uses a standard layout. At the top, a navigation bar allows users to access the other website sections such as Home, About Us, Login, and Register. The main content section displays a contact form that users can use to enter their name, email, phone number, and message. There is also a contact number provided for users to contact the developer directly. The footer indicates the services offered, company information and social media links, ensuring there are multiple ways to contact the developer. The "Contact Us" webpage provides users with a trust authority and inquiring engagement where applicable by allowing greater connectability through contact options.

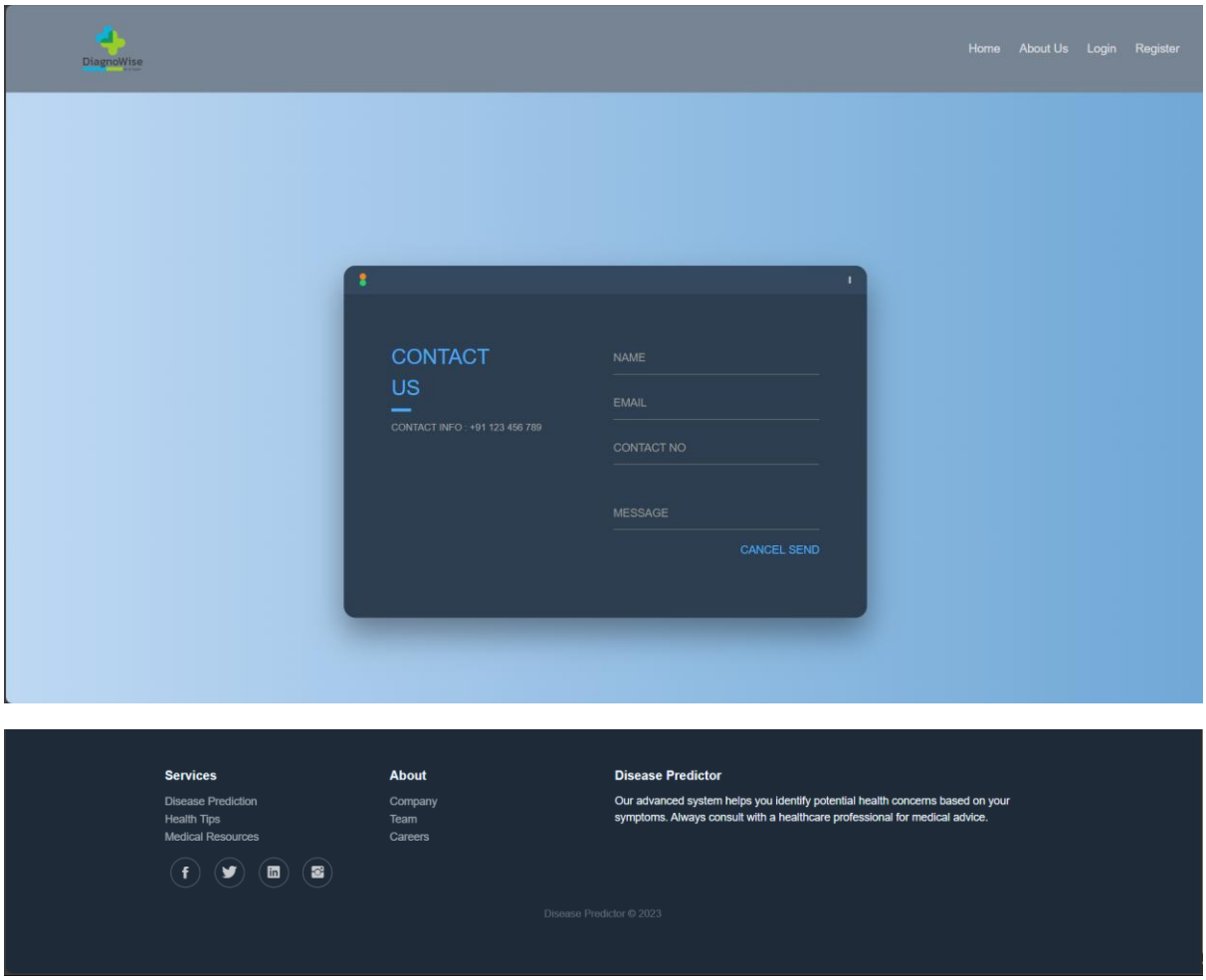


Fig 6.5 : Contact Page

6.2.5 About Page

This About Us page introduces the mission, vision and values of Disease Predictor web application. It clearly conveys the team's commitment to utilizing artificial intelligence for affordable visitor health assessment for everyone. Provides a synopsis of the platform functionality, willingness to evaluate user symptoms within a few seconds, and meaningful disease predictions via an easy and uncomplicated interface. The platform utilizes visual elements (cards, images) to convey information regarding the goal of the project, the reason users should trust its assessment and diagnosis, and the aim to provide a digital health care era. The navigation section also includes options for company, and contact information to simulate longer engagement of users.

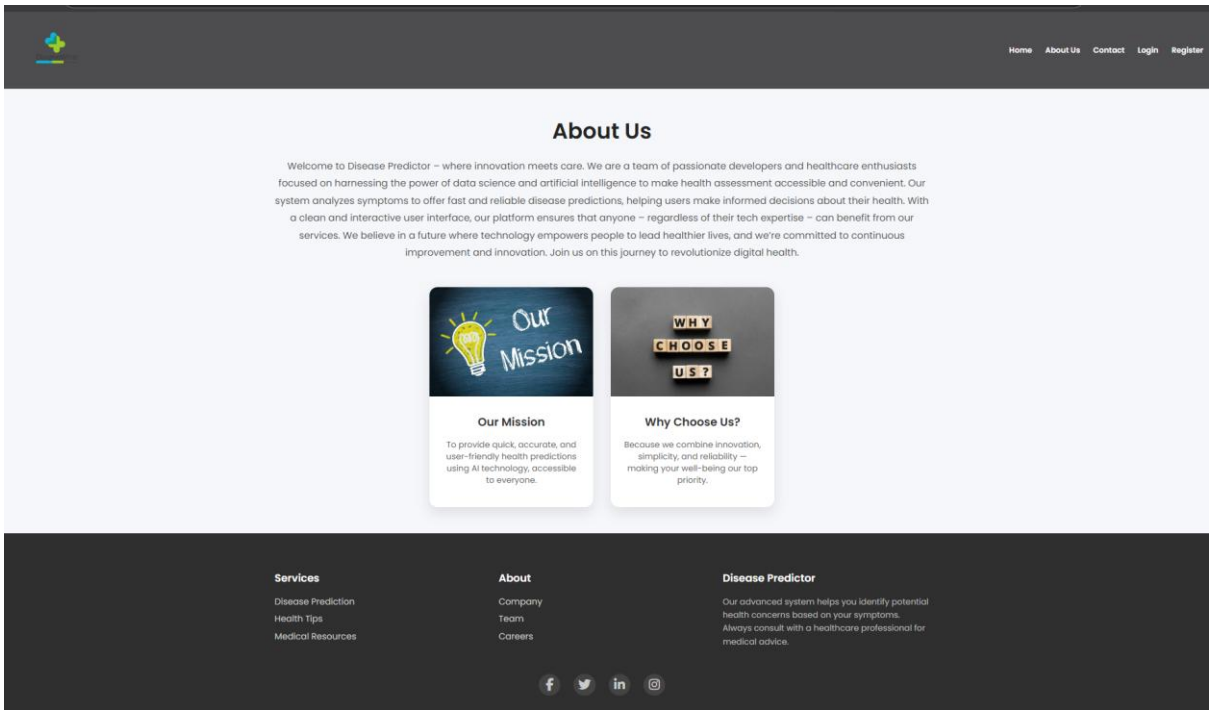


Fig 6.6 : About Page

6.2.6 Main Page

This webpage is an intelligent disease prediction service that allows users to input their symptoms in simple text and will produce a predicted diagnosis. The page is very clean & responsive. The layout is split between an image & branding of the app on one side and a simple, user-friendly form on the other side where the user enters their symptoms. Once they submit the information, the backend uses a predictive model to process the inputted symptoms and returns a probable disease result dynamically on the same webpage. The site also has good navigation to other sections in the site like Home, Contact, and About Us. The site has a modern footer with more resources, services and social media to explore. The objective of this tool is to assist users to make an informed decision in regards their health, by providing prologue information based on their symptoms.

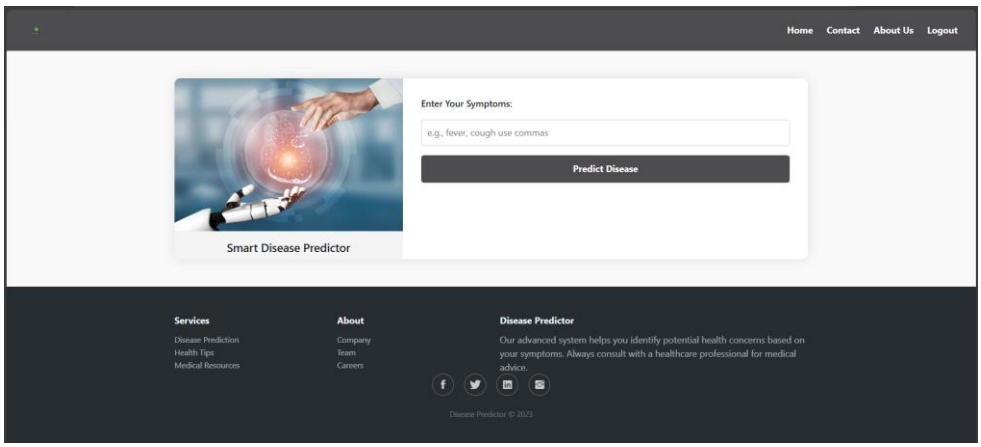


Fig 6.7 : Main page

CHAPTER 7

CODING AND IMPLEMENTATION

7.1 CODING

```
import sqlite3
import os
import pandas as pd
import joblib
from flask import Flask, request, render_template, jsonify,
redirect, url_for, session, flash
from werkzeug.security import generate_password_hash,
check_password_hash
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load and clean datasets
train_df = pd.read_csv("training.csv")
test_df = pd.read_csv("testing.csv")

# Remove unnamed columns like 'Unnamed: 133'
train_df = train_df.loc[:,
~train_df.columns.str.contains('^Unnamed')]
test_df = test_df.loc[:, ~test_df.columns.str.contains('^Unnamed')]

# Extract features and labels
X_train = train_df.iloc[:, 1:].apply(pd.to_numeric,
errors='coerce').fillna(0).astype(int)
y_train = train_df.iloc[:, 0]

X_test = test_df.iloc[:, 1:].apply(pd.to_numeric,
errors='coerce').fillna(0).astype(int)
y_test = test_df.iloc[:, 0]

# Initialize models
models = {
```

```
    "Naive Bayes": MultinomialNB(),
    "Random Forest": RandomForestClassifier(n_estimators=100,
random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=200)
}

# Train models and calculate accuracy
accuracies = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")

# Select best model
best_model_name = max(accuracies, key=accuracies.get)
best_model = models[best_model_name]
print(f"\n✅ Best Model: {best_model_name} with Accuracy:
{accuracies[best_model_name]:.4f}")

# Save best model
joblib.dump((X_train.columns.tolist(), best_model,
best_model_name), "best_disease_model.pkl")

# Load the model
symptoms_list, best_model, best_model_name =
joblib.load("best_disease_model.pkl")

# Load training data
training = pd.read_csv("Training.csv")
X = training.drop(columns=["prognosis"])
y = training["prognosis"]

# Store symptom list
```

```
symptoms_list = X.columns.tolist()

# Get list of disease names
disease_names = sorted(y.unique())

# Flask app
app = Flask(__name__)
app.secret_key = "your_secret_key"

# Initialize user DB
def init_user_db():
    if not os.path.exists("users.db"):
        conn = sqlite3.connect('users.db')
        conn.execute('CREATE TABLE users (id INTEGER PRIMARY KEY
AUTOINCREMENT, email TEXT UNIQUE, password TEXT)')
        conn.commit()
        conn.close()

init_user_db()

# Routes
@app.route("/", methods=["GET", "POST"])
def home():
    if "user" not in session:
        return redirect(url_for("welcome"))

    if request.method == "POST":
        user_symptoms = request.form["symptoms"].split(", ")
        print("User entered symptoms:", user_symptoms)

        # Ensure input vector has the same features as during
training
        input_vector = [1 if symptom in user_symptoms else 0 for
symptom in symptoms_list]
```

```
            conn.close()
            flash("Registration successful! Please login.")
            return redirect(url_for("login"))
        except sqlite3.IntegrityError:
            flash("Email already exists.")
            return redirect(url_for("register"))

    return render_template("Register.html")

@app.route("/logout")
def logout():
    session.pop("user", None)
    return redirect(url_for("login"))

@app.route("/welcome")
def welcome():
    return render_template("welcome.html")

@app.route("/contact")
def contact():
    return render_template("ContactUs.html")

@app.route("/about")
def about():
    return render_template("Aboutus.html")

@app.route("/healthdetails")
def healthdetails():
    return render_template("HealthDetails.html")

if __name__ == "__main__":
    app.run(debug=True, port=8000)
```

7.1.2 Main Page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>
  <title>DiagnoWise</title>
  <link href="https://fonts.googleapis.com/css?
family=Ubuntu:300,400,500,700" rel="stylesheet">
  <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.1/css/all.min.css">
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstr
ap.min.css" rel="stylesheet">
  <style>
    *, *:after, *:before {

      box-sizing: border-box;
    }
        body {
      background: #e5e8e8 ;
      color: #333;
      font-family: 'Ubuntu', sans-serif;
      position: relative;
         }
    .svg-container {
      position: absolute;
      top: 0;
      right: 0;
      left: 0;
      z-index: -1;
    }

    svg path {
```

```
    transition: .1s;
  }

  svg:hover path {
    d: path("M 800 300 Q 400 250 0 300 L 0 0 L 800 0 L 800 300
Z");
  }

  header {
    color: #fff;
    padding-top: 10vw;
    padding-bottom: 30vw;
    text-align: center;
    position: relative;
  }

  .auth-links {
    position: absolute;
    top: 20px;
    right: 40px;
  }

  .auth-links a {
    color: #fff;
    margin-left: 20px;
    font-weight: 500;
    text-decoration: none;
    font-size: 16px;
    transition: color 0.2s ease;
  }

  .auth-links a:hover {
    color: black;
  }
```

```
    h3 {
      font-weight: 400;
    }

    main {
      background: linear-gradient(to bottom,#e5e8e8
0%,#e5e8e8100%);
      border-bottom: 1px solid rgba(0, 0, 0, .2);
      position: relative;
      text-align: center;
      overflow: hidden;
      display: flex;
      justify-content: center;
      align-items: center;
    }

    main::after {
      border-right: 2px dashed #eee;
      content: '';
      position: absolute;
      top: calc(10vh + 1.618em);
      bottom: 0;
      left: 50%;
      width: 2px;
      height: 100%;
    }

    .main-nav a {
      color: #333;
      text-decoration: none;
      font-weight: 500;
      transition: color 0.2s ease;
    }

    .main-nav a:hover {
```

```
    @keyframes secFadeIn {
      0% {
        opacity: 0;
      }
      100% {
        opacity: 0.5;
      }
    }

    .card-text small {
      font-weight: bold;
      text-align: left;
      display: block;
      padding-left:90px;
    }



  </style>
</head>
<body>
  <div class="svg-container">
    <svg viewBox="0 0 800 400" class="svg">
      <path id="curve" fill="#50c6d8" d="M 800 300 Q 400 350 0 300
L 0 0 L 800 0 L 800 300 Z"></path>
    </svg>
  </div>

  <header>
  <div class="logo-container" style="position: absolute; top:
-30px; left: 2px; padding: 0px;">
    <img src="{{ url_for('static', filename='logo.png') }}"
```

```
  <header>
  <div class="logo-container" style="position: absolute; top:
-30px; left: 2px; padding: 0px;">
    <img src="{{ url_for('static', filename='logo.png') }}"
alt="Logo" style="height: 150px; width: auto;">
  </div>
    <div class="auth-links">
      <a href="{{ url_for('welcome') }}">Home</a>
<a href="{{ url_for('contact') }}">Contact Us</a>
      <a href="{{ url_for('about') }}">About Us</a>
      <a href="{{ url_for('login') }}">Login</a>
      <a href="{{ url_for('register') }}">Signup</a>
    </div>
    <div class="container">
      <div class="box">

          <div class="title">
              <span class="block"></span>
              <h1>Digno Wise<span></span></h1>
          </div>

          <div class="role">
              <div class="block"></div>
              <p>An AI EXPERT</p>
          </div>

      </div>
  </div>
  </header>

  <main>
    <div class="card mb-3" style="width: 75%; background-
color:#e5e8e8;">
      <div class="row g-0">
        <div class="col-md-4 d-flex flex-column align-items-center
```

```
justify-content-center">
          <img src="{{ url_for('static', filename='predi.png') }}"
class="img-fluid rounded-start mb-3" alt="..." style="height:
400px; width: 400px; object-fit: cover;">
          <h5 class="card-title text-center">Disease Prediction
Using Symptoms</h5>
        </div>
        <div class="col-md-8 d-flex flex-column justify-content-
center">
          <div class="card-body">
            <p class="">Get personalized disease predictions based
on your symptoms and health data.</p>
            <p class="card-text"><small class="text-muted">1.
Alwaysn consult with a doctor</small></p>
            <p class="card-text"><small class="text-muted">2. Not a
substitute for medical advice.</small></p>
            <p class="card-text"><small class="text-muted">3. Seek
medical guidance before use.</small></p>
            <p class="card-text"><small class="text-muted">4. Do
not take medication without prescription.</small></p>
            <p class="card-text"><small class="text-muted">5.
Consult a specialist if symptoms persist.</small></p>
            <p class="card-text"><small class="text-muted">6. Do
not self-diagnose or treat.</small></p>
            <p class="card-text"><small class="text-muted">7.
Medicine use must be doctor-approved.</small></p>
            <p class="card-text"><small class="text-muted">8.
Complete the full course of medicine.</small></p>
            <p class="card-text"><small class="text-muted">9.
Predictions may not be 100% accurate.</small></p>
            <p class="card-text"><small class="text-muted">10. Tool
results should be verified medically.</small></p>
          </div>
        </div>
      </div>
```

```
<footer class="footer">
  <div class="container">
    <div class="row">
      <div class="footer-col">
        <h4>company</h4>
        <ul>
          <li><a href="#">about us</a></li>
          <li><a href="#">our services</a></li>
          <li><a href="#">privacy policy</a></li>
          <li><a href="#">affiliate program</a></li>
        </ul>
      </div>
      <div class="footer-col">
        <h4>get help</h4>
        <ul>
          <li><a href="#">FAQ</a></li>
          <li><a href="#">shipping</a></li>
          <li><a href="#">returns</a></li>
          <li><a href="#">order status</a></li>
          <li><a href="#">payment options</a></li>
        </ul>
      </div>
      <div class="footer-col">
        <h4>online shop</h4>
        <ul>
          <li><a href="#">watch</a></li>
          <li><a href="#">bag</a></li>
          <li><a href="#">shoes</a></li>
          <li><a href="#">dress</a></li>
        </ul>
      </div>
      <div class="footer-col">
        <h4>follow us</h4>
```

```
      }

      window.addEventListener("scroll", function() {
        last_known_scroll_position = window.scrollY;

        if (!ticking) {
          window.requestAnimationFrame(function() {
            scrollEvent(last_known_scroll_position);
            ticking = false;
          });
        }

        ticking = true;
      });
    })();

    <script>
      const svgPath = document.getElementById('curve');
      document.querySelector('svg').addEventListener('mouseenter',
() => {
        svgPath.setAttribute("d", "M 800 300 Q 400 250 0 300 L 0 0
L 800 0 L 800 300 Z");
      });
      document.querySelector('svg').addEventListener('mouseleave',
() => {
        svgPath.setAttribute("d", "M 800 300 Q 400 350 0 300 L 0 0
L 800 0 L 800 300 Z");
      });
    </script>

  </script>
</body>
</html>
```

7.1.3 Health Insights Page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>
  <title>Health Insights</title>
  <link href="https://fonts.googleapis.com/css2?
family=Poppins:wght@400;600&display=swap" rel="stylesheet">
  <style>
    body {
      font-family: 'Poppins', sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f9f9;
      color: #333;
    }

    .logo {
      text-align: center;
    }

    .logo img {
      width: 100px;
    }


    .navbar-custom {
      background-color: #2c3e50;
      color: white;
      padding: 1rem 2rem;
      display: flex;
      justify-content: space-between;
      align-items: center;
    }
```

```
.navbar-custom h1 {
  margin: 0;
  font-size: 1.5rem;
}

.navbar-custom nav a {
  color: white;
  margin-left: 1rem;
  text-decoration: none;
  font-weight: 500;
}

header {
  background-color: #2c3e50;
  color: white;
  padding: 2rem;
  text-align: center;
}

header h1 {
  margin: 0;
  font-size: 2.5rem;
}

header p {
  margin-top: 0.5rem;
  font-size: 1.2rem;
}

.container {
  max-width: 1000px;
  margin: auto;
  padding: 2rem;
}
```

```
    .footer-custom ul li {
      margin: 0.3rem 0;
    }

    .footer-custom a {
      text-decoration: none;
      color: #ccc;
    }

    .footer-custom .social-icons {
      display: flex;
      justify-content: center;
      gap: 1.2rem;
      margin-top: 1rem;
    }

    .footer-custom .social-icons a {
      text-decoration: none;
      color: #ccc;
      font-size: 1.2rem;
    }

    .footer-custom p.copyright {
      margin-top: 1.5rem;
      font-size: 0.9rem;
      color: #aaa;
    }

    .navbar-custom nav a.active {
      border-bottom: 2px solid #27ae60;
    }
  </style>
</head>
<body>
```

```
  <!-- Navbar -->

  <div class="navbar-custom">
    <div class="logo" style="display: flex; align-items: center;">
    </div>
    <nav>
      <a href="/">Home</a>
      <a href="/predict">Predict</a>
      <a href="/about">About</a>
      <a href="/contact">Contact</a>
    </nav>
  </div>

  <!-- Header -->
  <header>
    <h1>Health Insights</h1>
    <p>Discover valuable health information tailored for you.</p>
  </header>

  <div class="container">

    <!-- 💾 General Health & Wellness -->
    <div class="section">
      <img src="static/General Health & Wellness.png" alt="General
Health & Wellness">
      <h2>💾 General Health & Wellness</h2>
      <p>
        Maintaining general health and wellness is essential for a
long and fulfilling life. Regular check-ups allow early detection
of potential illnesses, update necessary vaccinations, and ensure
continuous health monitoring. Your body’s vital signs—like blood
pressure, blood sugar, and BMI—are key indicators of well-being and
need routine observation. Understanding what these metrics
represent and how to interpret them helps you take proactive steps
to avoid serious health issues. Strengthening your immune system
```

```
natural defense against infections. Wellness isn't just about being
illness-free—it's about feeling your best every day. Small
lifestyle changes such as taking daily walks, reducing stress,
staying hydrated, and eating wholesome foods contribute
significantly to overall health. Make your wellness a priority by
staying informed and consistent with healthy routines.
      </p>
    </div>

    <!-- 🥗 Nutrition & Diet -->
    <div class="section">
      <img src="static/Nutrition & Diet.png" alt="Nutrition &
Diet">
      <h2>🥗 Nutrition & Diet</h2>
      <p>
        Proper nutrition forms the foundation of good health.
Superfoods such as berries, nuts, leafy greens, and fatty fish are
packed with vitamins, antioxidants, and essential nutrients that
boost immunity and reduce disease risk. A balanced diet ensures
your body receives carbohydrates, proteins, fats, and fiber in the
right proportions. Tailoring diets for different age groups is
vital—children need more calcium and iron, while older adults may
benefit from more fiber and less salt. Hydration also plays a
critical role in digestion, skin health, and energy. While the “8
glasses a day” rule is common, individual water needs vary
depending on lifestyle and climate. Nutritional awareness helps
prevent chronic conditions like diabetes, obesity, and heart
disease. Instead of focusing on calorie counting, focus on eating
whole, nutrient-dense foods, avoiding processed meals, and cooking
at home. Making informed choices now leads to better health
outcomes in the future.
      </p>
    </div>
```

```
    <!-- 💊 Health Tech & Innovation -->
    <div class="section">
      <img src="static/Health Tech & Innovation.png" alt="Health
Tech & Innovation">
      <h2>💊 Health Tech & Innovation</h2>
      <p>
        Technology is transforming healthcare at an unprecedented
pace. From AI-powered diagnostics to wearable fitness trackers,
patients can now monitor their health in real time. Symptom
checkers, telemedicine, and health apps provide accessibility like
never before. AI helps doctors analyze data, predict disease
outbreaks, and customize treatments for individuals. Wearables such
as smartwatches track heart rate, sleep, and activity levels,
encouraging users to stay engaged in their wellness journey. While
convenient, it’s important to use technology wisely and not replace
professional advice with apps. Tech should empower patients, not
confuse them. Future innovations like gene editing, remote
surgeries, and smart implants hold immense potential. As this field
grows, ethical concerns and data privacy must also be considered.
Health tech is not just about gadgets—it's about saving lives,
faster diagnoses, and personalized care.
      </p>
    </div>

    <!-- 🧍 Preventive Care -->
    <div class="section">
      <img src="static/Preventive Care.png" alt="Preventive Care">
      <h2>🧍 Preventive Care</h2>
      <p>
        Preventive care is your first line of defense against
disease. It includes screenings, vaccinations, lifestyle
counseling, and regular health evaluations. Getting vaccinated
protects you and those around you from serious illnesses like
measles, hepatitis, or HPV. Early detection of diseases, especially
cancers and chronic conditions, improves treatment success rates
```

```
cancers and chronic conditions, improves treatment success rates
and saves lives. Hand hygiene, clean environments, and avoiding
close contact with sick individuals are simple yet effective ways
to stop the spread of infections. Preventive care also includes
mental health checks, vision and dental assessments, and awareness
of hereditary risks. Staying educated and proactive in your health
decisions helps reduce future complications. Think of prevention as
investing in a healthier tomorrow. Don't wait for symptoms—take
control today.
      </p>
    </div>

  </div>

  <!-- Footer -->
  <footer class="footer-custom">
    <div class="footer-columns">
      <div class="footer-column">
        <h3>Services</h3>
        <ul>
          <li><a href="#">Disease Prediction</a></li>
          <li><a href="#">Health Tips</a></li>
          <li><a href="#">Medical Resources</a></li>
        </ul>
      </div>
      <div class="footer-column">
        <h3>About</h3>
        <ul>
          <li><a href="#">Company</a></li>
          <li><a href="#">Team</a></li>
          <li><a href="#">Careers</a></li>
        </ul>
      </div>
```

```
      <div class="footer-column">
        <h3>Disease Predictor</h3>
        <p>
          Our system helps identify health concerns based on
symptoms. Always consult with a healthcare professional.
        </p>
      </div>
    </div>
    <div class="social-icons">
      <a href="#"><img src="static/facebook.png" alt="Facebook"
width="24"/></a>
      <a href="#"><img src="static/twitter.png" alt="Twitter"
width="24"/></a>
      <a href="#"><img src="static/linkedin.png" alt="LinkedIn"
width="24"/></a>
      <a href="#"><img src="static/instagram.png" alt="Instagram"
width="24"/></a>
    </div>
    <p class="copyright">© 2025 Disease Predictor. All rights
reserved.</p>
  </footer>

</body>
</html>
```

7.2 IMPLEMENTATION

The implementation phase is one of the most important stages of the software development lifecycle. Implementation is the process of transforming the design, algorithms, and logic that has been defined into an actual system that works as intended through coding, integration of components, testing, and deployment. This section will provide a detailed description of the implementation process for our health-based prediction system using Flask as the backend framework, HTML, CSS, and JavaScript for the front end, and SQLite as a database. It should be noted that implementation is much more than writing code, it is about setting up environments, modules, API integration, and making sure the deployment works.

7.2.1 Setting Up the Development Environment

Before coding begins it will be necessary to set up a proper development environment and this involves:

- Installing Flask: The first step is to create a virtual environment using venv or virtualenv, then install flask using pip (pip install flask) to be able to create a web application using Python with a relatively light footprint or size and flexibility.
- Setting Up Sqlite: SQLite database is lightweight and is easily integrated with flask and when using the sqlite3 module, it is a great fit for smaller web applications.
- Installation of Required Packages: The application also requires additional Python packages such as flask_sqlalchemy, flask_wtf, flask_login, and so on, based on the user functionalities.

7.2.2 Developing the Backend Logic

The backend takes care of the core backend logic: authentication, storage, prediction, and forms.

User Registration/Login

User authentication is done primarily through flask_login. The following functions are implemented:

- register_user(): Receives form inputs, and hashes passwords with werkzeug.security before committing it to the user.db database.
- login_user(): Authenticates users by comparing data received to database data.
- logout_user(): Kills the session and logs the user out.

DataBase Connectivity

DataBase Models are created using Flask SQLAlchemy:

CRUD functionality is implemented using the SQLAlchemy methods for querying:

- db.session.add()
- db.session.commit()
- User.query.filter_by()

7.2.3 Developing the Frontend

Frontend development is done with HTML, CSS, and Bootstrap. JavaScript was used to add interactivity and utilize client-side validation of user input.

HTML Templates

HTML templates are stored in the templates folder and rendered using Flask's render_template() function. The used templates are:

- index.html: the initial landing page; an overview.
- register.html: the form to create a new user.
- login.html: the webpage for existing users to log in.
- dashboard.html: the user interface to access health prediction functionality.

Styling via CSS

Both custom CSS and classes provided by Bootstrap contribute to a polished layout. Important responsive design features are considered throughout development for usability across platforms.

JavaScript Integration

JavaScript is used for:

- Validation of forms (ensuring fields are not empty, validating field formats)
- Creating dynamically rendered modal popups
- Loading content dynamically, if necessary

7.2.4 Functionality Integration of Prediction Logic

The main function of the application is disease prediction based on user-entered symptoms. This is modeled as follows:

Symptom Input Form

Users make selections or enter symptoms by interacting with form fields. The form data is then sent to the back-end server in a request with POST.

Prediction Logic Algorithm

The training of the prediction logic was carried out offline with scikit-learn using datasets that contain symptoms and the corresponding diseases with which those symptoms are

associated. Scikit-learn produces a trained model that is saved as joblib or pickle properly as model.get_predict_data(). Since the trained model cannot sent and cannot not sent directly into the Flask app at the time it is produced, the saved model can only be loaded into the Flask app.

When running, the back end process the inputted data in the form, reshapes the data as needed, and pushes the inputted data to the model for prediction to occur. This prediction outcome is then provided to the user as intended.

Admin Panel

An in-app functionality scope needed to be implemented for managing users and data across that built application. An admin panel/dashboard functionality is implemented to manage users, data, and processes of the application. Administrators of the app need different capabilities to:

- Provide users information
- Delete entries if needed
- View prediction activity logged with timestamp across users

Errors and Validations

The system also includes error handlers for a number of exceptions, such as:

- 404 Error Page
- 500 Internal Server Error
- Input Validation Errors

When an error occurs, the user will see user-friendly messages.

7.2.7 Deployment and Hosting

It is important to complete deployment as part of implementation. The several steps are:

Deployment Platforms

- Local Server: Deployed on a local web sever for development and testing.
- Production: Deployed on a platform like Heroku, Render, or DigitalOcean.

Configuration Files

- Procfile for Heroku
- requirements.txt for Heroku
- runtime.txt for Heroku
- Using gunicorn as the WSGI HTTP Server.

7.2.8 Security

Security measures have been implemented at various levels to ensure the integrity of user information and system information:

- Password hashes: Passwords are hashed using Werkzeug.
- HTTPS setup: HTTPS setup is configurable as part of the deployment process.
- Session Management: Flask-login is used to manage secure sessions.
- Input Validation: Input sanitization avoids SQL injection attacks and XSS attacks.

7.2.9Testing During Implementation

Testing is done in parallel with overall implementation which can include three testing strategies:

- Unit Testing: Testing individual functions.
- Integration Testing: Testing modules that are working together.
- System Testing: Testing the application as a whole.

Automated and manual testing has been performed based on your degree of automation of the testing procedures to ensure that you are providing functionality and fixing of bugs.

7.2.10 Documentation and Maintenance

The implementation has been documented and includes:

- API Endpoints and documentation.
- Routes and mappings of URLs.
- Data Flow diagrams.

User guides and Admin instructions.

7.2.11 Future Enhancements

- The findings can be used to enhance the application which can include the following:
- Adding the training of ML models in the app for the model updates.
- Improved Admin analytics with visual dashboards.
- Real-time monitoring using IoT may be an area of future improvement.

During the implementation phase, the procedures are performed to turn the system design into a working product. Front-end and back-end technologies, secure login systems, predictive capabilities, and an intuitive user experience have been included in the health prediction web application. Every task related to the set-up, coding, testing, and deployment have been implemented throughout the phases to ensure quality and

reliability. Additionally, these tasks will create extensibility, scalability, and adherence to modern web standards.

FUTURE SCOPE OF THE PROJECT

The Smart Disease predictor project has significantly potential for future enhancements and real-world deployment. While the current version is a robust foundation, various improvements and extensions can be made to increase its accuracy, usability, and practical applicability.

Below are some of the key areas that can be explored in the future : Integration with Real-time Medical API's :

The System can be enhanced by integrating with medical api's such as those from WHO or Mayo Clinic to fetch updated disease and symptom data in real time, ensuring the system remains relevant and accurate.

Addition of More Complex Datasets :

Expanding the training dataset to include real-world medical records and multi-class symptoms can make predictions more precise and applicable to a broad range of user data.

History Tracking :

Monitoring health Trends over time, and receive personalized health suggestions based on past inputs.

Prescription and Treatment Suggestions :

The System can be expanded to provide generic treatment advice or over-the-counter medicine recommendations, which can be cross-checked by a medical practitioner.

Multilingual Support :

To increase accessibility, especially in rural or diverse areas, the application support multiple languages and dialects, making it more inclusive.

REFERENCES

1. John D., et al. (2015). *Early Detection of Diabetes Using Logistic Regression*. International Journal of Medical Informatics.
2. Smith A., & Zhang Y. (2016). *Wearable Technology for Real-Time Health Monitoring*. IEEE Transactions on Biomedical Engineering.
3. Patel K., et al. (2017). *Comparative Study of Machine Learning Algorithms for Heart Disease Detection*. International Journal of Computer Applications.
4. Gupta R., & Sharma M. (2018). *Blockchain-Based Secure Transfer of Medical Records*. Journal of Health Informatics.
5. Lee J., & Thomas E. (2018). *Mobile App Usage Among Chronic Illness Patients*. mHealth Journal.
6. Anderson L., et al. (2019). *Deep Learning for Lung Disease Classification*. Computers in Biology and Medicine.
7. Kaur S., & Verma P. (2019). *Challenges of EHR Integration in Indian Healthcare Systems*. Indian Journal of Medical Informatics.
8. Chen H., & Yu L. (2020). *Natural Language Processing in Symptom Checker Applications*. AI in Healthcare.
9. Kumar S., et al. (2020). *Survey on IoT Devices in Rural Health Monitoring*. Journal of Internet of Medical Things.
10. Singh A., & Kapoor N. (2020). *Chatbot-Based Healthcare Assistant Implementation*. Advances in AI and Medicine.
11. Davis B., et al. (2021). *Predictive Analytics for Hospital Readmissions*. Journal of Healthcare Engineering.
12. Raj M., & Joshi A. (2021). *Stress Detection Using Machine Learning and Wearables*. Proceedings of IEEE EMBS.
13. Lin C., & Zhao H. (2021). *Cloud-Based Medical Records with Biometric Security*. Health IT Journal.
14. Ahmed N., et al. (2022). *COVID-19 Prediction Using Real-Time Symptom Logging*. Pandemic Informatics Journal.
15. Banerjee T., & Sinha P. (2022). *User Perception of Telemedicine Post COVID-19*. Journal of Telehealth and Telecare.
16. George M., et al. (2022). *AI-Based Triage Systems in Emergency Departments*. IEEE Transactions on Health Informatics.

17. Yadav A., & Desai R. (2023). *Hypertension Risk Prediction Using ML*. Health Analytics Review.

18. Wang X., & Liu D. (2023). *ECG Signal Analysis Using CNN-LSTM Hybrid*. Bioinformatics and Signal Processing Journal.

19. Sharma L., & Nair V. (2023). *Privacy-Preserving Frameworks for Wearable Devices*. Security in Healthcare Systems.

20. Thomas K., et al. (2024). *Scalable Flask-Based Health Prediction Systems*. Journal of Web-Based Applications.

21. Government of India. (2000). *Information Technology Act*. https://www.indiacode.nic.in

22. European Union. (2016). *General Data Protection Regulation (GDPR)*. https://gdpr.eu

23. U.S. Department of Health & Human Services. (1996). *HIPAA Regulations*. https://www.hhs.gov/hipaa

24. OpenAI. (2024). *Understanding AI Ethics and Safety*. https://openai.com

25. W3C. (2023). *Web Content Accessibility Guidelines (WCAG)*. https://www.w3.org/WAI/standards-guidelines/wcag

BIBILIOGRAPHY

1. Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer.
2. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
3. Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.
4. Tan, P. N., Steinbach, M., & Kumar, V. (2019). *Introduction to Data Mining*. Pearson.
5. Kelleher, J. D., Mac Carthy, M., & Korvir, B. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics*. MIT Press.
6. Kumar, V., & Minz, S. (2020). *Healthcare Systems and Informatics: Applications of AI in Health*. Elsevier.
7. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
8. Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.
9. Tiwari, P., & Shukla, R. (2021). *Flask Web Development Projects*. Packt Publishing.
10. Dietrich, D. et al. (2015). *Internet of Things: Principles and Paradigms*. Elsevier.
11. European Union. (2016). *General Data Protection Regulation (GDPR)*. Retrieved from https://gdpr.eu
12. Government of India. (2000). *Information Technology Act*. Retrieved from https://www.indiacode.nic.in
13. U.S. Department of Health and Human Services. (1996). *HIPAA Privacy Rule*. Retrieved from https://www.hhs.gov/hipaa
14. OpenAI. (2024). *AI Ethics and Safety Guidelines*. Retrieved from https://openai.com
15. W3C. (2023). *Web Content Accessibility Guidelines (WCAG)*. Retrieved from https://www.w3.org/WAI/standards-guidelines/wcag