HEALTH MONITOR

A PROJECT REPORT

submitted by

GEORGE KURIAN (MUT22CS065)

JOMAL SANISH (MUT22CS077)

RAHUL K (MUT22CS119)

SONIA SARA JOSEPH (MUT22CS143)

to

the APJ Abdul Kalam Technological University in partial fullfilment of the requirements for the award of the Degree

of

Bachelor of Technology

In

Computer Science & Engineering

Department of Computer Science & Engineering

Muthoot Institute of Technology and Science

Varikoli PO, Puthencruz - 682308

APRIL 2025

DECLARATION

We hereby declare that this submission is our own work and that, to the best of our knowledge and belief, it contains no material previously written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

GEORGE KURIAN (MUT22CS065)

JOMAL SANISH (MUT22CS077)

RAHUL K (MUT22CS119)

SONIA SARA JOSEPH (MUT22CS143)

Place: MITS Varikoli

Date: 9 April 2025

Muthoot
Institute of Technology & Science

CERTIFICATE

This is to certify that the report entitled **"HEALTH MONITOR"**, *submitted by* **GEORGE KURIAN, JOMAL SANISH, RAHUL K** *and* **SONIA SARA JOSEPH** *to Muthoot Institute of Technology and Science, Varikoli for the award of the degree of Bachelor of Technology in Computer Science & Engineering is a bonafide record of the project work carried out by them, under our supervision and guidance. The content of the report, in full or parts have not been submitted to any other Institute or University for the award of any other degree or diploma.*

Ms.Sheena K.V	Ms.Jisha James	Dr.Anishin Raj M M
Project Guide	Project Coordinator	Head of the Department

Place: MITS Varikoli

Date: 9 April 2025

ACKNOWLEDGMENT

We are grateful to almighty who has blessed us with good health, committed and continuous interest throughout the project work.

We express our sincere thanks to our guide, **Ms.Sheena K.V**, Assistant Professor, Department of Computer Science And Engineering, Muthoot Institute of Technology and Science and **Dr.Anishin Raj M M**, Professor, Head Of the Department, Muthoot Institute of Technology and Science for their guidance and support which were instrumental in all the stages of the project work and without whom the project could not have been accomplished.

We are grateful to our project coordinators **Ms.Jisha James and Ms.Sneha Sreedevi** Assistant Professors, Department of Computer Science And Engineering, Muthoot Institute of Technology and Science, for their guidance and support.

We would like to thank **Dr. Neelakantan P.C.**, Principal, Muthoot Institute of Technology and Science, Varikoli for providing us all the necessary facilities.

The last but not the least, we extend our sincere thanks to the entire teaching and non-teaching staff of Computer Science And Engineering of Muthoot Institute of Technology and Science for their help and co-operation throughout our project work.

ABSTRACT

This project aims to develop an AI-powered website that helps users analyze their medical reports, such as blood test results and provides clear, easy-to-understand analysis. The system will use machine learning techniques to process uploaded PDF reports, identify key health metrics, and detect potential problems. It will offer recommendations on the type of doctor to consult (e.g., cardiologist, General Physician) and suggest possible remedies or lifestyle changes based on the findings. The platform will prioritize user-friendliness and data security, providing helpful insights while emphasizing that its recommendations complement, not replace, professional medical advice. This platform seeks to empower users with preliminary medical insights, streamline the decision-making process, and encourage timely medical consultations, ultimately improving healthcare accessibility and outcomes.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Health Monitor project leverages artificial intelligence to transform the way individuals access and interpret complex medical data. By integrating predictive models for medical report analysis into a user-friendly web application, we empower users to gain clear, actionable health insights from their test results and symptom inputs. This project report will walk you through the design, development, and implementation of our AI-powered platform, highlighting the code architecture, model training processes, and the interactive interface that bridges the gap between technical diagnostics and patient understanding.

1.2 SCOPE AND MOTIVATION

In today's fast-paced world, access to timely and accurate medical insights is critical. With advancements in artificial intelligence (AI), there is a significant opportunity to revolutionize healthcare by providing users with tools that analyze medical reports and offer actionable recommendations. Many individuals receive complex medical test results but lack the expertise to interpret them effectively. This gap often delays necessary medical consultations, potentially exacerbating health issues. The proposed AI-powered medical analysis and recommendation platform aims to bridge this gap, empowering users with accessible, comprehensible, and actionable medical insights.

CHAPTER 2

PROPOSED WORK

2.1 OBJECTIVES

1. Medical Report Analysis - The platform analyzes user-submitted symptoms and lab test reports to detect possible health conditions. It uses predictive analytics to assess risks and suggests likely diagnoses in simple, easy-to-understand language.
2. Disease Prediction and Detection - Using machine learning, the system predicts potential diseases based on symptoms and test values. It provides risk levels—such as low, medium, or high—and offers basic treatment suggestions or next steps, like seeing a specialist.
3. User Authentication and Medical History - Secure login and role-based access protect user data and personalize the experience. The platform also saves medical history to support ongoing health tracking and more accurate assessments over time.
4 Web Accessibility and Usability - The platform works on smartphones, tablets, and desktops. It has a simple interface for entering data, uploading reports, and viewing results. Users can choose Auto Mode for general checks or Specific Mode for focused analysis.
5. Scalability and Future Expansion - The system can grow to include more diseases and conditions. New AI models can be added to improve predictions and expand features.

2.2 PROBLEM STATEMENT

Patients often struggle to interpret complex medical test results, leading to delayed care, which creates a pressing need for an accessible, AI-powered platform that accurately analyzes diverse medical data and translates it into clear, actionable insights for timely intervention.

2.3 EXISTING SYSTEM AND PROPOSED SOLUTION

2.3.1 EXISTING SYSTEM

Existing healthcare platforms often lack the ability to analyse medical test reports or provide personalised, data-driven insights. Many rely on static symptom checkers and offer limited integration of user data, resulting in vague or generic recommendations. Additionally, complex interfaces and minimal use of advanced AI reduce accessibility and effectiveness, making timely and accurate guidance difficult for users.

2.3.2 PROPOSED SYSTEM

- Medical Report Interpretation : AI models convert lab data into simple, clear explanations. Users get easy-to-understand summaries of their results.
- Symptom and Data Integration : Symptoms and lab reports are analyzed together using machine learning. The system offers a unified and accurate health overview.
- Predictive Insights and Disease Detection : Predictive models assess disease risks and classify them as low, medium, or high. Users receive personalized suggestions and recommended next steps.
- Security and Health History Tracking : Secure login and role-based access protect user data. Medical history is saved for continuous monitoring and improved future analysis.

CHAPTER 3

PROJECT DESIGN

3.1 SYSTEM ARCHITECTURE

Client Interaction

The user uploads symptoms and lab reports through the web app. They can choose between Auto Mode or Specific Mode for analysis.

Frontend (UI Element)

Handles user login, history, report upload, and mode selection. Sends user inputs to the backend and displays the results.

Backend (Flask API)

Manages authentication, data processing, and model selection. Coordinates with data processing tools and returns results to the UI.

Data Processing and Storage

Gemini LLM: Generates natural language explanations.

Specific Model: Predicts diseases based on test data.

Document Parser: Extracts data from uploaded reports.

MongoDB: Stores user data and medical history.

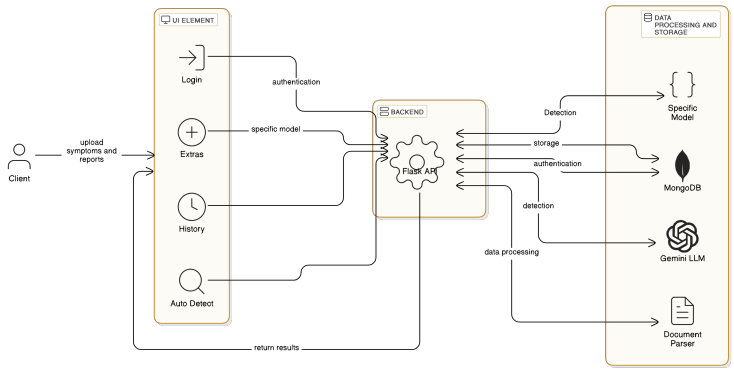

Figure 3.1: System Architecture

3.2 MODULES

3.2.1 Input and Data Collection Module

This module is responsible for gathering user inputs such as symptoms and lab test values, either manually or through uploaded medical reports.

Components: User Interface Form – Allows users to enter symptoms and upload lab reports.

Document Parser – Extracts data from uploaded files (PDFs, text).

Functionality: Captures and organizes patient data in a structured format. Parses lab report content to extract relevant values and parameters for analysis.

3.2.2 Analysis and Prediction Module

This module performs the core analysis using AI models to assess user health data, predict possible diseases, and generate insights.

Components: Machine Learning Models – Predict disease risks based on input data.

Gemini LLM – Generates user-friendly explanations of the results.

Flask Backend – Manages communication between frontend and AI models.

Functionality: Analyzes symptoms and test results to identify potential health risks. Assigns risk levels (low, medium, high) and generates diagnosis suggestions. Converts complex data into understandable language using LLM.

3.2.3 Results and Interaction Module

This module communicates the analysis results to the user clearly and interactively, providing recommendations and tracking history.

Components: Web Dashboard – Displays results, risk levels, and suggested actions.

History Log – Stores past reports and results for user reference.

Functionality: Shows disease predictions, risk scores, and next-step advice. Allows users to view past assessments and monitor trends over time. Supports Auto Mode for general checkups and Specific Mode for targeted conditions.

3.3 DATA FLOW DIAGRAM

3.3.1 DFD LEVEL 0

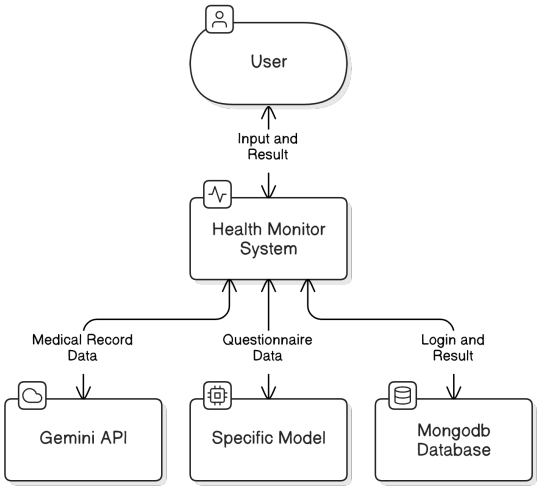

Figure 3.2: DFD LEVEL 0

3.3.2 DFD LEVEL 1

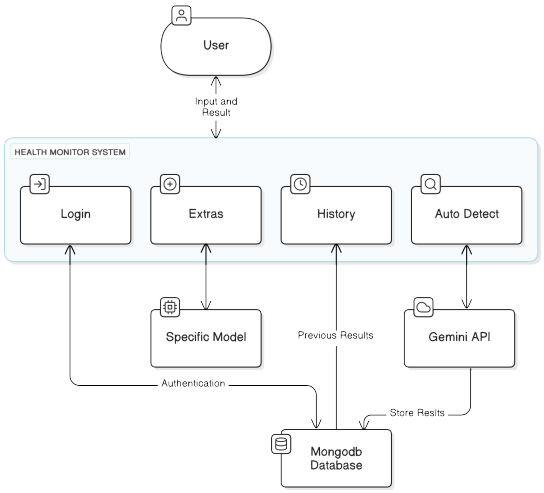

Figure 3.3: DFD LEVEL 1

3.3.3 DFD LEVEL 2

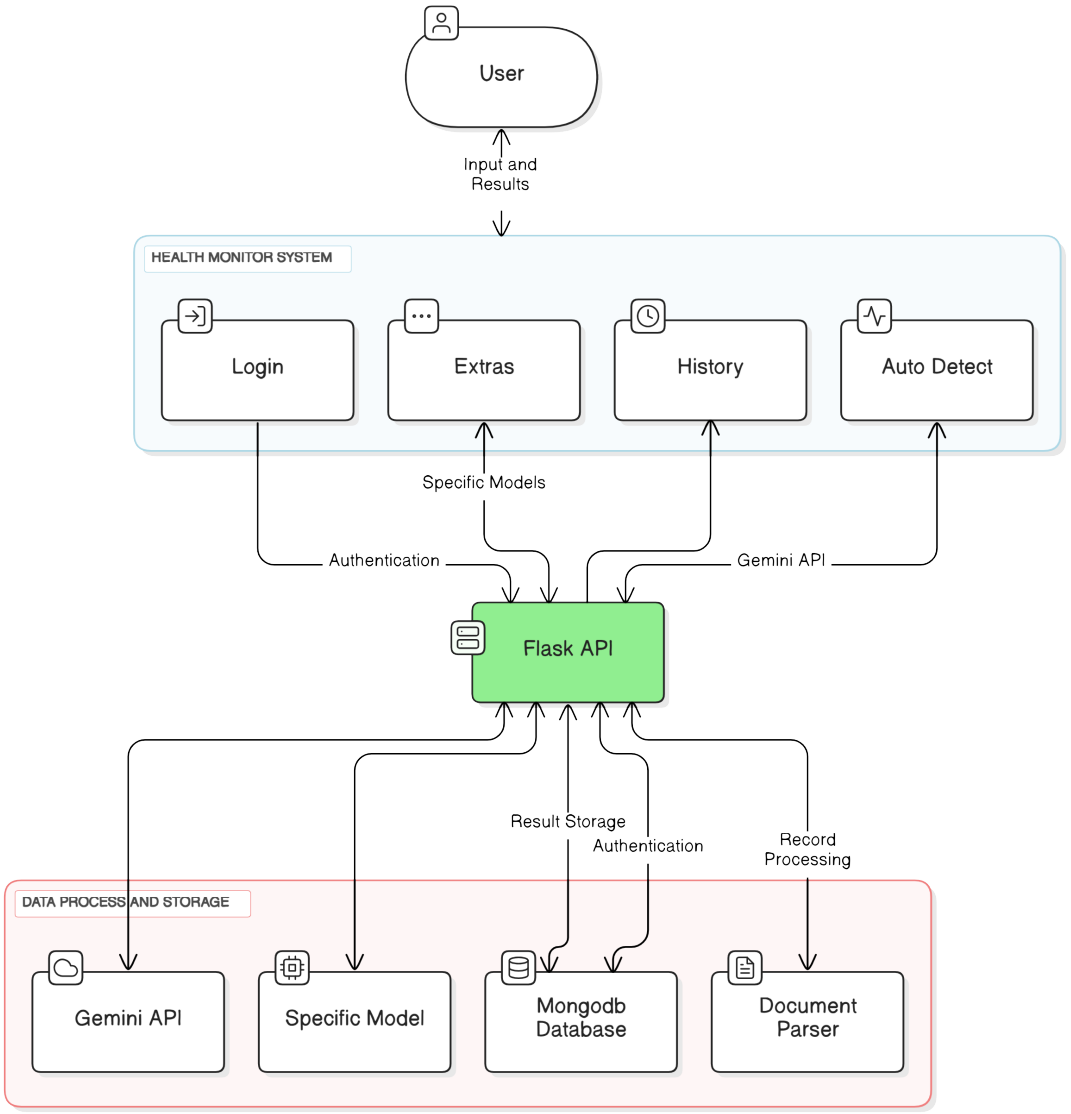

Figure 3.4: DFD LEVEL 2

3.4 DATABASE TABLE DESIGN

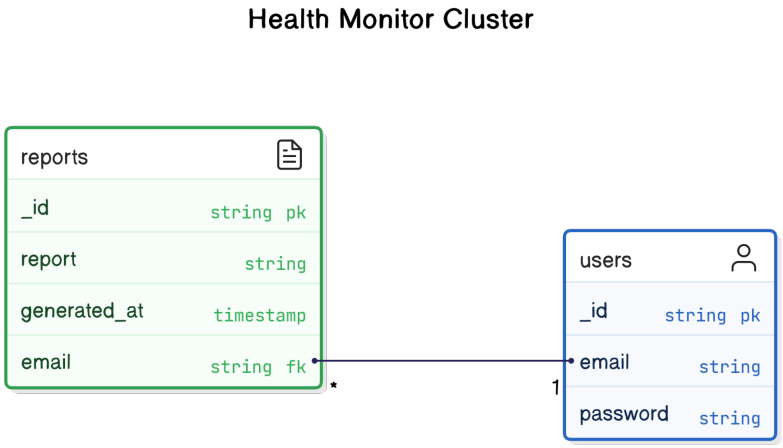

Figure 3.5: Database Architecture

3.5 GUI DESIGN

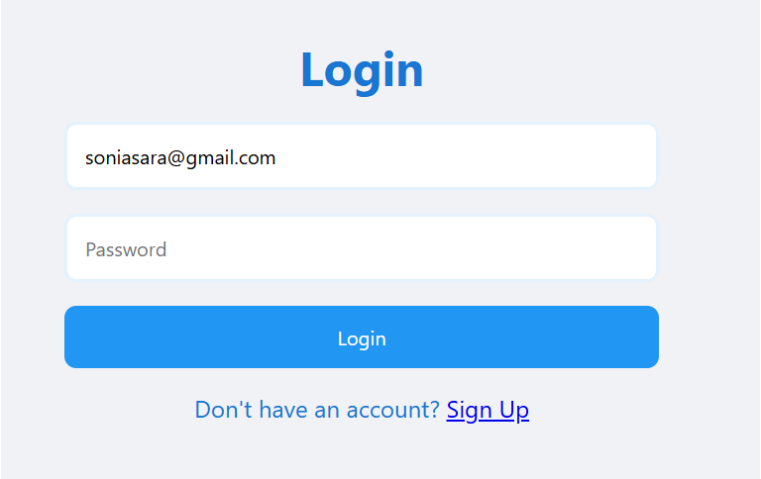

Figure 3.6: Login Page

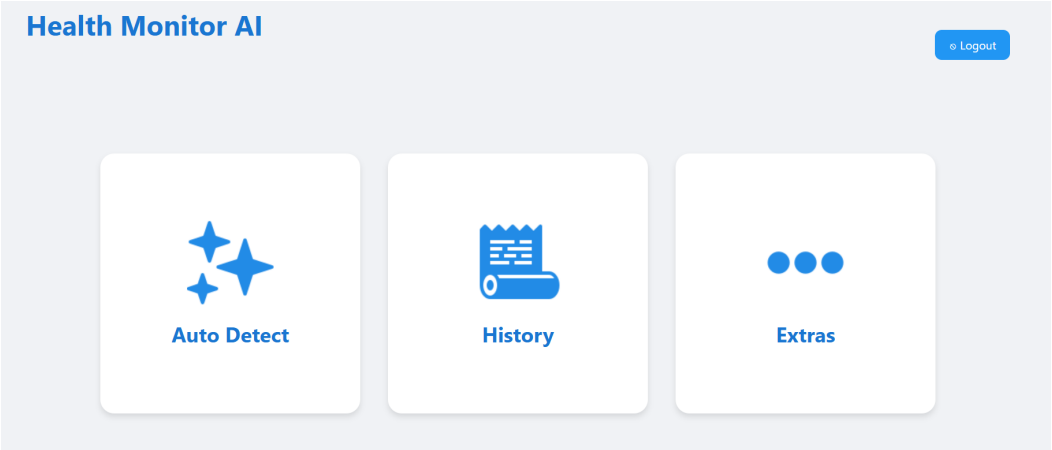

Figure 3.7: Home Screen

3.6 TECHNOLOGY STACK

3.6.1 Hardware Stack

Processor: Modern multi-core CPU (e.g., Intel i5/i7, AMD Ryzen 5/7) for development and server hosting.

RAM: Minimum 8 GB for backend and model execution; 4 GB for client-side use.

Storage: At least 256 GB SSD to store datasets, reports, and model files.

Display: Devices with a resolution of 1024×768 or higher for optimal UI rendering.

Networking: Stable internet connection for real-time predictions and smooth user experience.

GPU: Not required (models are lightweight and CPU-compatible).

3.6.2 Software Stack

Frontend

HTML, CSS, JavaScript – Basic web technologies for building the user interface.

Tailwind CSS / Bootstrap – For fast and clean UI styling.

Backend

Python with Flask – Server-side logic and API development.

RESTful API – Enables communication between frontend and backend.

Machine Learning and Data Handling

scikit-learn – Used for training and running prediction models.

Pandas and NumPy – Data manipulation and numerical operations.

SimpleImputer – Handles missing data in inputs.

PyPDF2 / python-docx – Parses lab reports in PDF and Word format.

Database

MongoDB (Cloud) – Stores user information, medical history, and prediction results.

Authentication and Security

JWT (JSON Web Tokens) – Secures user sessions and data access.

bcrypt / Passlib – Ensures password encryption and secure login.

Development Tools

VS Code / PyCharm – Code editors used for writing and debugging.

Thunder Client – For testing APIs during development.

Git – Version control for managing source code and collaboration.

Browser Support

Chrome, Firefox, Edge, Safari – Modern browsers required for accessing the web app.

3.7 SYSTEM REQUIREMENTS

Hardware Requirements:

Processor: A modern multi-core processor such as Intel i5/i7 or AMD Ryzen 5/7 is recommended. It ensures smooth backend operations and model execution, while users only need a processor capable of running a web browser efficiently.

Memory (RAM): At least 8 GB of RAM is ideal for handling data processing, model predictions, and backend services. For basic usage and frontend access, 4 GB of RAM is sufficient.

Storage: A minimum of 256 GB SSD is recommended to store datasets, reports, model files, and application code. SSD storage also ensures faster access and better performance.

GPU: A dedicated GPU is not required for this project. The models used are lightweight

and run efficiently on standard CPUs.

Display: A screen resolution of 1024×768 or higher is recommended. This ensures a clear and user-friendly experience when interacting with the web application.

Networking: A stable internet connection is necessary for accessing the application and receiving real-time predictions. It also ensures secure and uninterrupted communication between the client and server.

Software Requirements:

Operating System: The project supports development and usage on Windows, macOS, and Linux. The web application is accessible on any device with a modern OS, including Android and iOS.

Programming Language: The core application is developed in Python, chosen for its simplicity, readability, and strong ecosystem for AI and data science.

IDE / Code Editor: Development is done using popular editors like Visual Studio Code or PyCharm, which provide robust tools for Python development and debugging.

Libraries and Frameworks

Flask: Lightweight framework used for building the backend and handling API routes.

scikit-learn: Used for training and running machine learning models for disease prediction.

Pandas and NumPy: Handle data manipulation and numerical operations efficiently.

SimpleImputer (from scikit-learn): Manages missing values in the dataset.

PyPDF2 and python-docx: Used to extract medical data from PDF and Word reports.

Web Technologies: Frontend is built using HTML, CSS, and JavaScript to ensure a responsive and user-friendly interface.

Web Browser: A modern browser such as Chrome, Firefox, Edge, or Safari is required with JavaScript enabled to access and use the application effectively.

Database: A cloud-hosted MongoDB is used to store user profiles, medical history, and prediction results. It offers flexibility and scalability for document-based storage.

Version Control: Git is used for tracking code changes, version control, and collaboration among developers through GitHub.

CHAPTER 4

IMPLEMENTATION

4.1 CODE SNIPPETS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Health Monitor AI</title>
  <link rel="stylesheet" href="styles.css" />
</head>

<body>
  <!-- Login Page -->
  <div id="login-page" class="container auth-container">
    <h1>Login</h1>
    <form id="login-form">
      <input type="email" id="login-email" placeholder="Email" required />
      <input type="password" id="login-password" placeholder="Password" required />
      <button type="submit">Login</button>
    </form>
    <p>Don't have an account? <a href="#" onclick="showSignUp()">Sign Up</a></p>
  </div>

  <!-- Sign Up Page -->
  <div id="signup-page" class="container auth-container hidden">
    <h1>Sign Up</h1>
    <form id="signup-form">
      <input type="email" id="signup-email" placeholder="Email" required />
      <input type="password" id="signup-password" placeholder="Password" required />
      <button type="submit">Sign Up</button>
    </form>
    <p>Already have an account? <a href="#" onclick="showLogin()">Login</a></p>
  </div>

  <!-- Home Page -->
  <div id="home-page" class="container hidden">
    <div class="top-nav">
      <h1 class="main-title">Health Monitor AI</h1>
      <button class="logout-button" onclick="logout()">⎋ Logout</button>
    </div>
    <div class="home-container">
      <div class="option-box" onclick="navigateTo('auto-detect')">
        <img
          src="" />
        <h2>Auto Detect</h2>
      </div>
```

Figure 4.1: Frontend 1

```
    <!-- History option -->
    <div class="option-box" onclick="navigateTo('history')">
      <img
        src="" />
      <h2>History</h2>
    </div>
    <div class="option-box" onclick="window.location.href='extras.html'">
      <img src="" />
      <h2>Extras</h2>
    </div>
  </div>
</div>

<!-- Auto Detect Page -->
<div id="auto-detect-page" class="container hidden">
  <button class="nav-button" onclick="goBack()">← Back</button>
  <div class="auto-detect-container">
    <h1>Auto Detection</h1>
    <div class="input-section">
      <textarea placeholder="Enter your symptoms here..."></textarea>
      <div class="file-upload">
        <p>Or upload medical reports</p>
        <input type="file" accept=".pdf,.doc,.docx,.txt" />
      </div>
      <button class="submit-button" onclick="viewResults()">View Results</button>
    </div>
  </div>
</div>
<!-- History Page -->
<div id="history-page" class="container hidden">
  <button class="nav-button" onclick="goBack()">← Back</button>
  <h1>History Reports</h1>
  <div id="reports-container" class="reports-grid">
  </div>
</div>
<script src="script.js"></script>
<script>
  function logout() {
    localStorage.removeItem('userEmail');
    document.querySelectorAll('.container').forEach(el => el.classList.add('hidden'));
    document.getElementById('login-page').classList.remove('hidden');
  }
</script>
</body>
```

Figure 4.2: Frontend 2

```python
from flask import Flask, request, jsonify # backend to frontend connectivity
from flask_cors import CORS
from pymongo import MongoClient #mongodb
from google import genai  # Gemini AI
import PyPDF2  # for parsing pdf
from docx import Document  # for parsing document format
from datetime import datetime # current time
import pickle  # for loading our ML model

app = Flask(__name__)
CORS(app)

# Connect to MongoDB Cluster
mongo_client = MongoClient("mongodb+srv://22cs029:mits123@primarycluster.eu4op.mongodb.net/?appName=PrimaryCluster")
db = mongo_client.get_database("HealthMonitor")
users_collection = db.get_collection("users")
reports_collection = db.get_collection("reports")

# Initialize the Gemini AI client
client = genai.Client(api_key="AIzaSyCygULILuxUr_lPWYxXY2mFqcOQnpWeuyU")

def extract_text_from_file(file):
    if file.filename.endswith('.pdf'):
        pdf_reader = PyPDF2.PdfReader(file)
        text = '\n'.join([page.extract_text() for page in pdf_reader.pages])
        return text
    elif file.filename.endswith(('.doc', '.docx')):
        doc = Document(file)
        return '\n'.join([para.text for para in doc.paragraphs])
    elif file.filename.endswith('.txt'):
        return file.read().decode('utf-8')
    return None

@app.route('/signup', methods=['POST'])
def signup():
    data = request.get_json()
    email = data.get("email")
    password = data.get("password")
    if not email or not password:
        return jsonify({"success": False, "error": "Email and password are required"}), 400

    if users_collection.find_one({"email": email}):
        return jsonify({"success": False, "error": "User already exists"}), 400

    users_collection.insert_one({"email": email, "password": password})
    return jsonify({"success": True}), 200
```

Figure 4.3: Backend 1

```python
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    email = data.get("email")
    password = data.get("password")
    if not email or not password:
        return jsonify({"success": False, "error": "Email and password are required"}), 400

    user = users_collection.find_one({"email": email})
    if user and user.get("password") == password:
        return jsonify({"success": True}), 200
    return jsonify({"success": False, "error": "Invalid credentials"}), 401

@app.route('/analyze', methods=['POST'])
def analyze():
    try:
        text_input = request.form.get('text')
        file = request.files.get('file')
        email = request.form.get('email')

        input_text = text_input or ''
        if file:
            extracted_text = extract_text_from_file(file)
            input_text += '\n' + (extracted_text if extracted_text else '')

        prompt = f"""Analyze these medical inputs and provide:
Don't use numbers or asterisks; keep the response short.
1. Possible diseases (list top 3 with likelihood percentage)
2. Urgency level (low/medium/high)
3. Whether to consult a doctor (yes/no)
4. Recommended specialist type
5. Brief explanation
6. Keep the result as if it's coming from a medical practitioner
7. Do not use numbers or asterisks
8. Disclaimer: The analysis is AI-generated; please consult a doctor for detailed evaluation.

Medical Inputs:
{input_text}
"""

        response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=prompt
        )

        now = datetime.now()
```

Figure 4.4: Backend 2

```
        header = f"Report generated on {current_date}\n"
        full_report = header + response.text

        reports_collection.insert_one({
            "report": full_report,
            "generated_at": now,
            "email": email
        })

        return jsonify({
            'success': True,
            'analysis': full_report
        })

    except Exception as e:
        return jsonify({
            'success': False,
            'error': str(e)
        })

@app.route('/predict_diabetes', methods=['POST'])
def predict_diabetes():
    try:
        data = request.get_json()
        # Extract and convert input features from JSON
        features = [
            int(data.get("age")),
            1 if data.get("gender").strip().lower() == "male" else 0,
            int(data.get("polyuria")),
            int(data.get("polydipsia")),
            int(data.get("sudden_weight_loss")),
            int(data.get("weakness")),
            int(data.get("polyphagia")),
            int(data.get("genital_thrush")),
            int(data.get("visual_blurring")),
            int(data.get("itching")),
            int(data.get("irritability")),
            int(data.get("delayed_healing")),
            int(data.get("partial_paresis")),
            int(data.get("muscle_stiffness")),
            int(data.get("alopecia")),
            int(data.get("obesity"))
        ]
        # Load the pre-trained diabetes model
        with open("Models/diabetes_model.pkl", "rb") as f:
            model = pickle.load(f)
        prediction = model.predict([features])
```

Figure 4.5: Backend 3

```
        model = pickle.load(f)
        prediction = model.predict([features])
        result = "Diabetic" if prediction[0] == 1 else "Non-Diabetic"
        disclaimer = ("Disclaimer: The prediction is AI-generated and should not be considered a medical diagnosis. "
                      "Please consult a doctor for a detailed evaluation.")
        return jsonify({"success": True, "prediction": result, "disclaimer": disclaimer})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)})

@app.route('/predict_liver', methods=['POST'])
def predict_liver():
    try:
        data = request.get_json()
        # Extract liver disease features; convert appropriate types.
        # Expected features (in order):
        # Age, Sex, Total_Bilirubin, Direct_Bilirubin, Alkaline_Phosphotase,
        # Alamine_Aminotransferase, Aspartate_Aminotransferase, Total_Proteins,
        # Albumin, Albumin_and_Globulin_Ratio
        features = [
            float(data.get("age")),
            1 if data.get("gender").strip().lower() == "male" else 0,
            float(data.get("total_bilirubin")),
            float(data.get("direct_bilirubin")),
            float(data.get("alkaline_phosphotase")),
            float(data.get("alt")),
            float(data.get("ast")),
            float(data.get("total_proteins")),
            float(data.get("albumin")),
            float(data.get("ag_ratio"))
        ]
        with open("Models/liver_model.pkl", "rb") as f:
            liver_model = pickle.load(f)
        prediction = liver_model.predict([features])
        # Depending on your dataset, adjust prediction meaning.
        result = "Liver Disease Detected" if prediction[0] == 1 else "No Liver Disease Detected"
        disclaimer = ("Disclaimer: The prediction is AI-generated and should not be considered a medical diagnosis. "
                      "Please consult a doctor for a detailed evaluation.")
        return jsonify({"success": True, "prediction": result, "disclaimer": disclaimer})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)})

@app.route('/predict_mental', methods=['POST'])
def predict_mental():
    try:
        data = request.get_json()
        # Convert inputs to numeric features using the same logic as in training
        gender_num = 1 if data.get("gender", "").strip().lower() == "male" else 0
```

Figure 4.6: Backend 4

```
        data = request.get_json()
        # Convert inputs to numeric features using the same logic as in training
        gender_num = 1 if data.get("gender", "").strip().lower() == "male" else 0
        age_val = float(data.get("age"))
        year_val = float(data.get("year"))
        cgpa_map = {
            "2.50 - 2.99": 0,
            "3.00 - 3.49": 1,
            "3.50 - 4.00": 2
        }
        cgpa_val = cgpa_map.get(data.get("cgpa", "").strip(), 1)  # default to 1 if not found
        marital_val = 1 if data.get("marital", "").strip().lower() == "yes" else 0
        specialist_val = 1 if data.get("specialist", "").strip().lower() == "yes" else 0

        # Use the same features as in training
        features = [gender_num, age_val, year_val, cgpa_val, marital_val, specialist_val]

        with open("Models/mental_model.pkl", "rb") as f:
            mental_model = pickle.load(f)

        prediction = mental_model.predict([features])[0]
        result = "Likely to have mental health issues" if prediction == 1 else "Less likely to have mental health issues"
        disclaimer = ("Disclaimer: The prediction is AI-generated and should not be considered a medical diagnosis. "
                      "Please consult a mental health professional for a detailed evaluation.")
        return jsonify({"success": True, "prediction": result, "disclaimer": disclaimer})
    except Exception as e:
        return jsonify({"success": False, "error": str(e)})

@app.route('/history', methods=['GET'])
def history():
    try:
        email = request.args.get("email")
        if not email:
            return jsonify({"success": False, "error": "Email is required"}), 400

        reports = list(reports_collection.find({"email": email}).sort("generated_at", -1))
        for r in reports:
            r["_id"] = str(r["_id"])
            r["generated_at"] = r["generated_at"].strftime("%Y-%m-%d %H:%M:%S")
        return jsonify({"success": True, "reports": reports}), 200
    except Exception as e:
        return jsonify({"success": False, "error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 4.7: Backend 5

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
import pickle

# Load the dataset (adjust path and delimiter if necessary)
df = pd.read_csv("liver_disease.csv")  # assuming comma-separated

# Preprocess data:
# Use the correct column names. In your CSV, the gender column is "Gender".
df["Gender"] = df["Gender"].apply(lambda x: 1 if x.strip().lower() == "male" else 0)

# (Optional) Rename columns if needed; e.g., if there's a spelling mistake:
df = df.rename(columns={"Total_Protiens": "Total_Proteins"})

# Define the features based on your CSV columns
features = [
    "Age",
    "Gender",
    "Total_Bilirubin",
    "Direct_Bilirubin",
    "Alkaline_Phosphotase",
    "Alamine_Aminotransferase",
    "Aspartate_Aminotransferase",
    "Total_Proteins",
    "Albumin",
    "Albumin_and_Globulin_Ratio"
]

X = df[features]
y = df["Dataset"]  # Adjust if your target column name is different

# Impute missing values in X using the median of each column
imputer = SimpleImputer(strategy="median")
X_imputed = imputer.fit_transform(X)

# Train the logistic regression model using the imputed data
model = LogisticRegression(max_iter=1000)
model.fit(X_imputed, y)

# Save the trained model to a pickle file
with open("liver_model.pkl", "wb") as f:
    pickle.dump(model, f)

print("Liver disease model trained and saved as liver_model.pkl")
```

Figure 4.8: Model Training

4.2 SCREENSHOTS

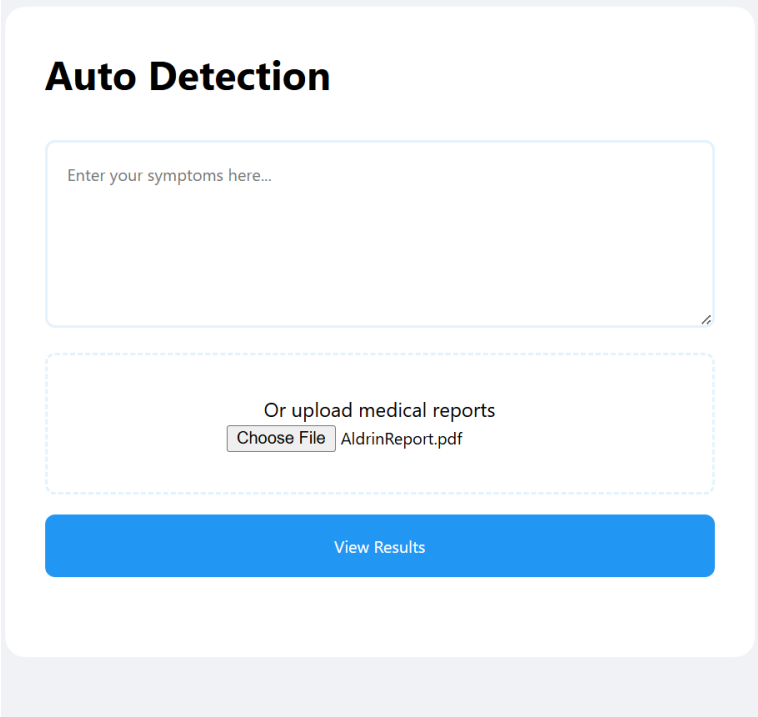

Figure 4.9: Auto Detect Page

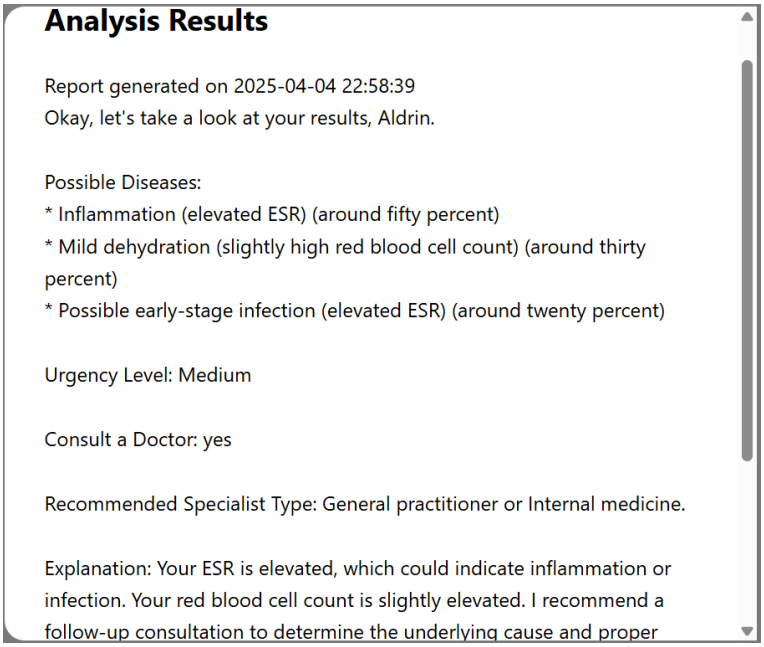

Figure 4.10: Analysis Result

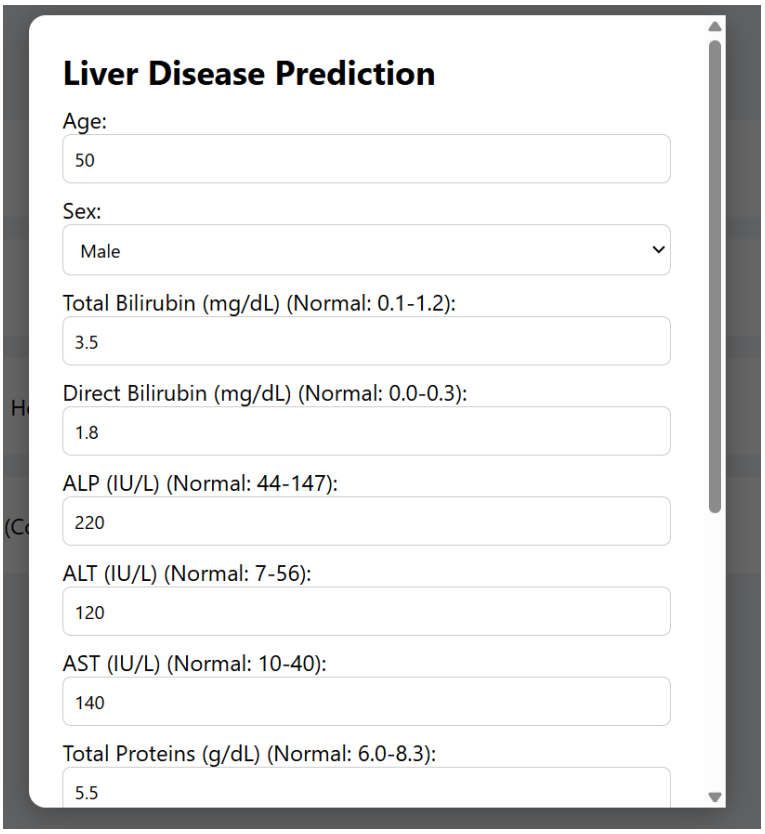

Figure 4.11: Specific Disease

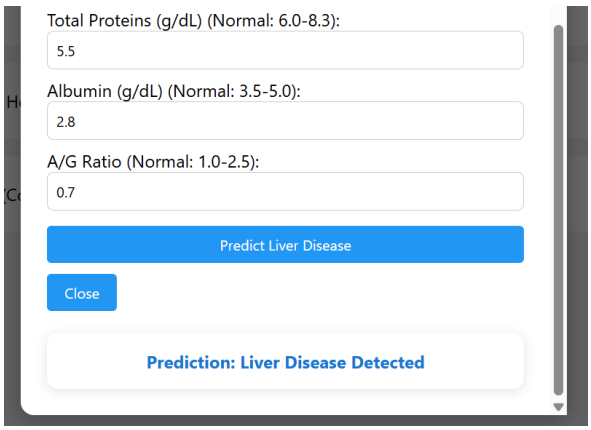

Figure 4.12: Prediction Result

CHAPTER 5

CONCLUSION

Health Monitor is an AI-powered medical platform designed to make healthcare insights more accessible and understandable for everyday users. It allows individuals to input symptoms and lab test results, which are then analyzed using advanced machine learning models trained on verified medical data. The system predicts possible health conditions, evaluates risk levels, and provides clear, personalized recommendations in simple language.

With a responsive web interface, secure user authentication, and features like Auto Mode and Specific Mode, the platform ensures ease of use and flexibility. It also maintains a record of user health history, supporting long-term tracking and better health management over time.

Validated against expert medical assessments, Health Monitor has demonstrated high reliability and accuracy. It addresses a critical gap in healthcare by transforming technical diagnostic information into actionable insights that users can understand and act upon. In doing so, it empowers users to make informed decisions, seek timely medical attention, and take proactive control of their well-being.

The system's modular and scalable architecture also allows for future expansion, including support for more diseases, integration with wearable health devices, and enhanced AI capabilities. Health Monitor stands as a step forward in combining technology with patient-centered care to support more transparent, efficient, and proactive healthcare delivery.

CHAPTER 6

REFERENCES

1. World Health Organization (2021). Health systems strengthening: Challenges and opportunities.
2. Smith, J. and Johnson, P. (2022) The Impact of AI on Medical Data Analysis: A Case Study of IBM Watson Health. Journal of Health Informatics, 45, 145-156.
3. Healthcare (Basel). 2022 Mar 15;10(3):541. Machine-Learning-Based Disease Diagnosis: A Comprehensive Review
4. UC Irvine Machine Learning Repository.
5. Health-LLM: Personalized Retrieval-Augmented Disease
6. Prediction System by Mingyu Jin, Qinkai Yu
7. Gemini API Documentation
8. Mongodb Documentation
9. Flask Documentation
10. Scikit-learn Documentation

APPENDICES

Figure 6.1: Slide 1

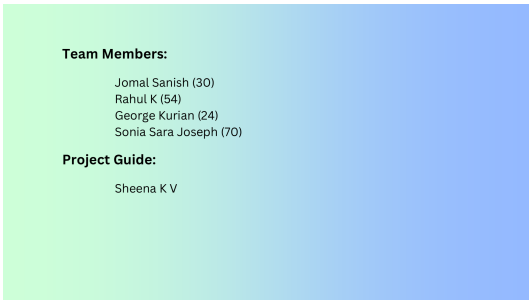

Figure 6.2: Slide 2

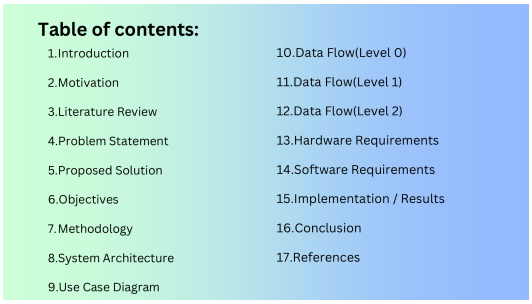

Figure 6.3: Slide 3

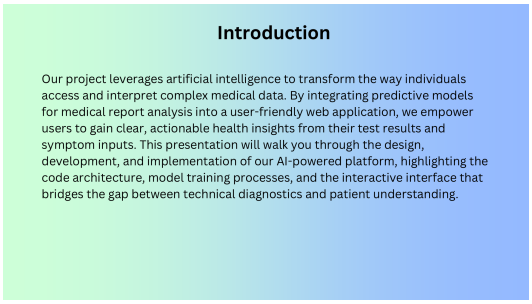

Figure 6.4: Slide 4

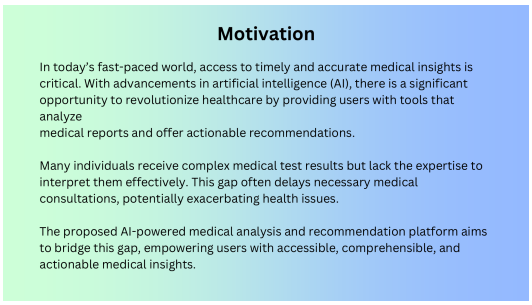

Figure 6.5: Slide 5

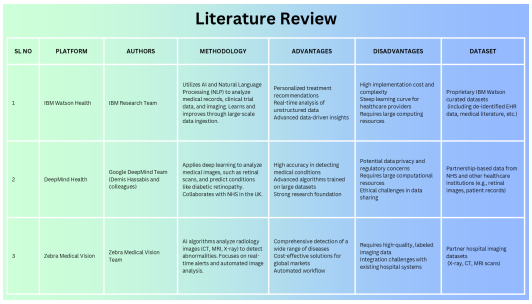

Figure 6.6: Slide 6

SL NO	PLATFORM	AUTHORS	METHODOLOGY	ADVANTAGES	DISADVANTAGES	DATASET
4	Aidoc	Aidoc Medical Team	AI analyzes medical images in real time, flagging critical findings and triaging patients for faster diagnosis and treatment.	Real-time alerts and improved triage Automated workflow reduces clinician burden Faster turnaround for imaging results	Integration with hospital PACS/RIS systems can be complex Reliance on consistent imaging protocols	Multiple partner hospital data (radiology images from various modalities)
5	PathAI	PathAI Research Team	Machine learning assists pathologists in analyzing pathology slides for cancer diagnosis, grading, and more.	High diagnostic accuracy Provides pathologists with interpretive insights Reduces variability in pathology readings	Relies on high-quality digitized slides May require specialized hardware for scanning Regulatory hurdles	Digital pathology slides from partner labs, research institutions
6	Babylon Health	Babylon Health Team	AI-driven chatbot analyzes symptoms and provides preliminary health advice. Integrates with advanced telemedicine and remote monitoring tools.	24/7 accessibility Convenience for patients, reduced waiting times Wide user reach with incremental improvements	Accuracy depends on user input and dataset diversity Regulatory and licensing challenges Potential biases	Proprietary patient data (collected via user app interactions and telemedicine services)

Figure 6.7: Slide 7

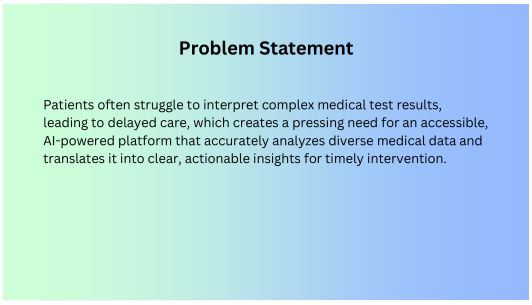

Figure 6.8: Slide 8

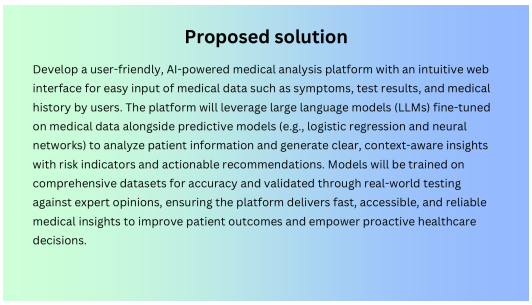

Figure 6.9: Slide 9

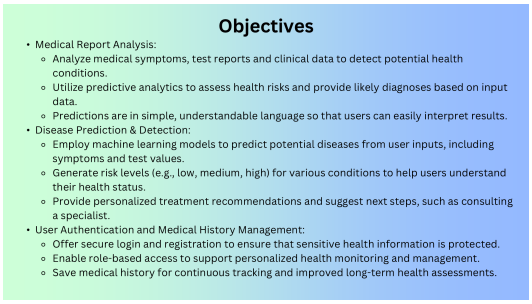

Figure 6.10: Slide 10

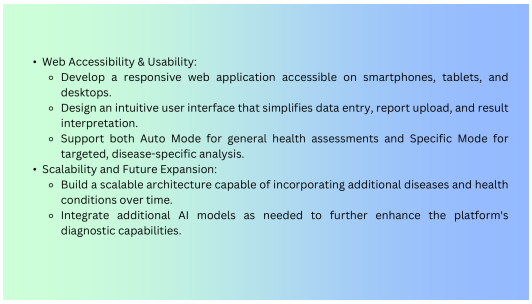

Figure 6.11: Slide 11

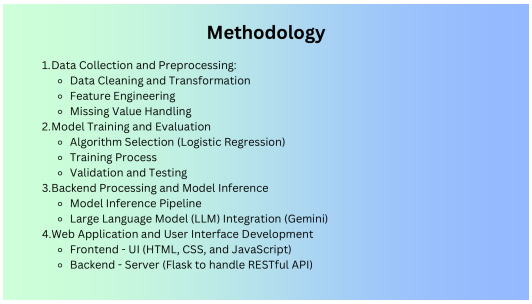

Figure 6.12: Slide 12

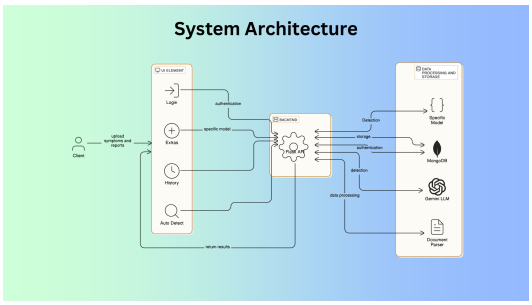

Figure 6.13: Slide 13

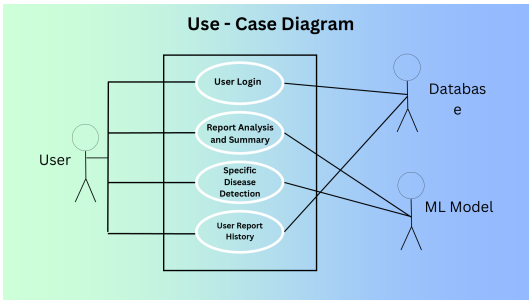

Figure 6.14: Slide 14

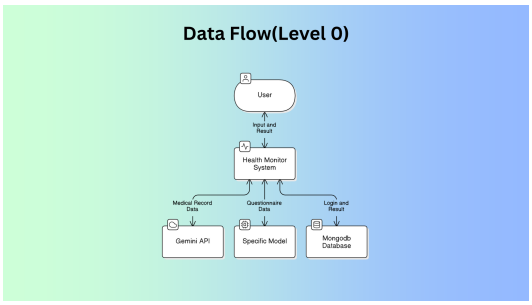

Figure 6.15: Slide 15

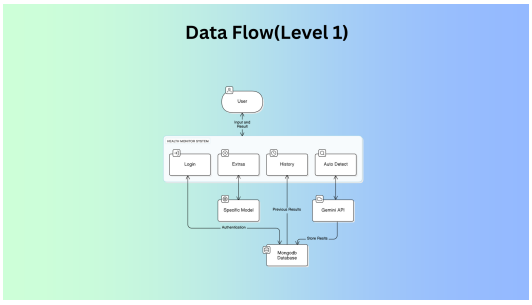

Figure 6.16: Slide 16

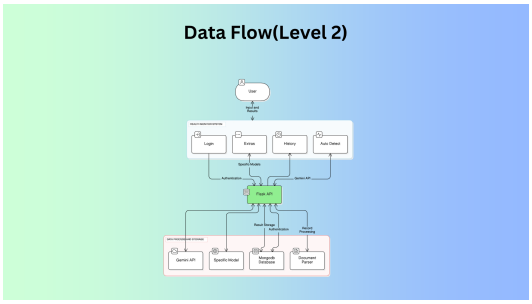

Figure 6.17: Slide 17

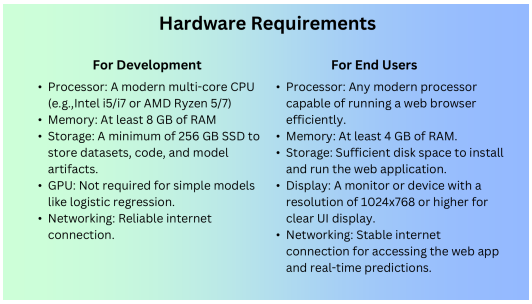

Figure 6.18: Slide 18

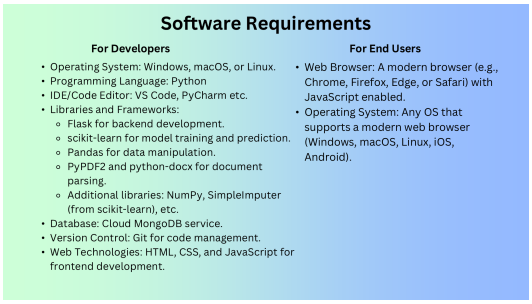

Figure 6.19: Slide 19

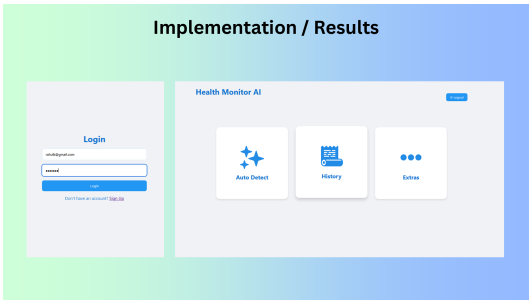

Figure 6.20: Slide 20

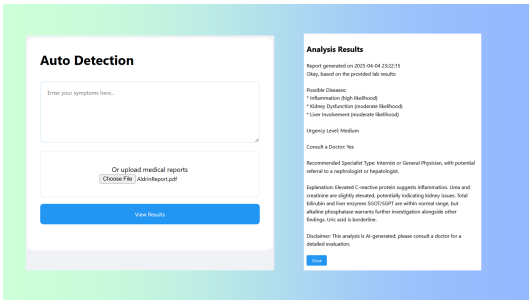

Figure 6.21: Slide 21

Figure 6.22: Slide 22

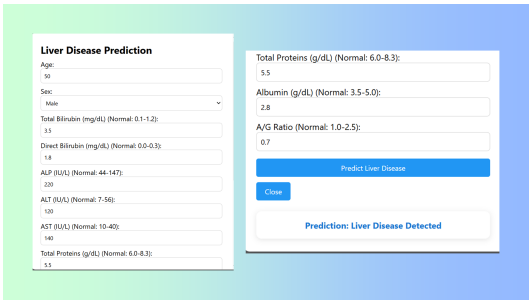

Figure 6.23: Slide 23

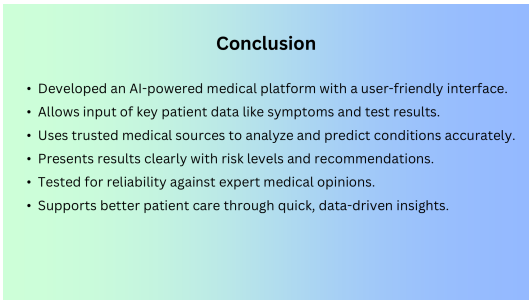

Figure 6.24: Slide 24

Figure 6.25: Slide 25

Figure 6.26: Slide 26