# Internship Report

*Submitted in partial fulfilment of the requirement for the award of*

*Degree of*

**BACHELOR OF COMPUTER APPLICATION**

GUIDE:

**Karthikeyan C (External)**

**Trainer VCodez**

**Dr. Garima Tyagi (Internal)**

**Professor**

SUBMITTED BY:

**Om Singh Chauhan (K24404)**


## SCHOOL OF COMPUTER APPLICATION AND TECHNOLOGY

## CAREER POINT UNIVERSITY, KOTA

Colloquium-II (CAC771)

Aug - Dec, 2025

VCODEZ™
INNOVATING IDEAS

**09th October 2025**

**Subject: Internship Experience Letter**

**TO WHOMSOEVER IT MAY CONCERN**

This is to certify that **Mr. Om Singh Chauhan (Reg. No. K24404)** from **Career Point University, Rajasthan** in the Department of **BCA, (Data Science)** successfully completed an internship with Vcodez as **Data Science** in our company from August 2025 to October 2025.

During the internship, he has gained several learnings and developed considerable skills, including hands-on experience on Real time projects.

Besides showing high comprehension capacity, managing assignments with the utmost expertise, and exhibiting maximal efficiency, he has also maintained an outstanding professional presence and showcased excellent moral character throughout the internship period.

I hereby certify that overall work as excellent to the best of my knowledge. Wishing him the best of luck in future endeavors.

**For VCODEZ**

**Vishnu S**
**Manager – Human Resources**

**VCODEZ**
#51, Rattha Tek Meadows, Level 3, Tower C,
OMR, Sholinganallur, Chennai - 600119
+91 96001 89201 info@vcodez.com
www.vcodez.com

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Career Point University, Kota**, for providing me the opportunity to work on this project **titled *"MedGuardian: AI-Powered Early Disease Detection System."***

I extend my heartfelt thanks to my **External Guide, Mr. Karthikeyan C (VCodez Innovation Ideas),** for his valuable guidance, continuous support, and encouragement throughout the project. His expertise and insights played a significant role in shaping the technical components of this system.

I am also deeply grateful to my **Internal Guide, Dr. Garima Tyagi,** for her constant motivation, constructive feedback, and continuous support during the development and documentation phases of the project.

I would like to thank all the faculty members of the **School of Computer Application and Technology,** for providing a supportive learning environment that helped me enhance my technical and professional skills.

Finally, I express my heartfelt gratitude to my parents, classmates, and friends for their encouragement, inspiration, and support throughout my project journey.

**Om Singh Chauhan**

**Place : Kota**

**Date :**

# DECLARATION

I hereby declare that this Project Report titled **"MedGuardian: AI-Powered Early Disease Detection System",** submitted by me and approved by my project guide at the School of Computer Application and Technology (SOCA), Career Point University, Kota, is a bona fide work carried out solely by me.

I further declare that this project has **not been submitted** to any other University or Institution for the award of any degree, diploma, or certificate, nor has it been published previously.

| | | |
|---|---|---|
| **Project Name :** | ***MedGuardian: AI-Powered Early Disease Detection System*** | |
| **Student Name:** | Om Singh Chauhan | Signature |
| **Project Guide: (External)** | Mr. Karthikeyan C | Signature |
| **Project Guide: (Internal)** | Dr. Garima Tyagi | Signature |

# Brief About the Company

VCodez is a leading software solutions and technology consulting company committed to delivering innovative, scalable, and high-quality digital solutions for businesses across diverse sectors. Situated in Chennai, Tamil Nadu, VCodez has established itself as a trusted partner for organizations seeking to modernize operations, improve efficiency, and embrace digital transformation.

The company operates with a philosophy that technology is not just a tool—but a catalyst that empowers businesses to grow, adapt, and excel in a highly competitive digital world. With a strong emphasis on innovation, the team at VCodez consists of experienced developers, engineers, designers, and consultants who collaborate closely with clients to understand their unique requirements and translate ideas into practical, impactful solutions.

VCodez specializes in custom software development, mobile application development, AI and Machine Learning integration, edge computing solutions, IT consultancy, and cloud-based services. By focusing on the needs of both startups and established enterprises, the company ensures that every solution is tailored for performance, security, and long-term scalability. Their approach is client-centric, emphasizing transparency, continuous communication, and end-to-end support throughout the project lifecycle.

One of the key strengths of VCodez lies in its commitment to quality and its ability to adopt emerging technologies. Whether it is developing advanced AI-driven systems, building robust enterprise applications, or offering strategic IT consulting, the company strives to deliver solutions that add measurable value and support the client's long-term vision. This dedication has helped VCodez build strong, lasting relationships with clients, positioning the organization as a trusted technology partner in the industry.

**Mission**

The mission of VCodez is to empower businesses through innovative, scalable, and high-quality software solutions that drive growth, enhance operational efficiency, and improve customer experiences. The company aims to consistently deliver exceptional value through advanced technologies, domain expertise, and a deeply client-focused approach.

**Vision**

VCodez envisions becoming a global leader in the software solutions and digital transformation space, recognized for driving innovation, enabling intelligent systems, and fostering long-term partnerships. The company strives to create a future where technology delivers seamless, secure, and efficient digital ecosystems that elevate businesses across all industries.

**Corporate Details**

**Company Name**: VCodez
**Address**: #51, Rattha Tek Meadows, Level 3, Tower C, OMR, Sholinganallur, Chennai, Tamil Nadu – 600119
**Contact:** +91 9600189201
**Email**: info@vcodez.com
**Website:** www.vcodez.com

**Services Offered**

- Custom Software Development
- Mobile Application Development
- Edge Computing
- AI & Machine Learning Integration
- Cloud & IT Consultancy
- Technical Support & Maintenance

# Table of Contents



## Appendix

## A: Data Flow Diagram (DFD)

## B: Entity Relationship Diagram (ERD)

## C: Use Case Diagram (UCD)

## D: Data Dictionary (DD)

## E: Screen Shots

# 1 Project Title

MedGuardian: AI-Powered Early Disease Detection System

# 2 Problem Statement

Early detection of diseases remains one of the biggest challenges in modern healthcare. Most medical conditions are diagnosed late, leading to:

- Higher treatment costs
- Lower survival chances
- Increased burden on healthcare systems
- Dependency on manual diagnosis
- Limited availability of specialists

Early detection of diseases is a major challenge in modern healthcare systems. Many medical conditions, including chronic and lifestyle-related diseases, are often diagnosed in later stages, leading to increased medical costs, reduced survival rates, and delays in treatment. Traditional diagnosis also relies heavily on manual evaluation, which may be time-consuming and prone to human error. Additionally, the shortage of specialists limits accessibility to timely and accurate diagnosis.

Therefore, there is a strong need for an intelligent, automated, and accurate disease prediction system that assists doctors and patients by providing early warnings and risk assessments.

The goal of this project is to design MedGuardian, an AI-powered system that predicts the likelihood of diseases by analyzing patient health metrics such as age, glucose levels, blood pressure, symptoms, and lifestyle factors.

# 3 Project Description

MedGuardian is an AI-based healthcare application designed to support early disease identification using machine learning algorithms. It processes patient information, analyzes critical health indicators, and predicts the probability of potential diseases.

The system is capable of assessing risks for commonly diagnosed conditions such as diabetes, heart diseases, and liver diseases, and can be extended to other medical conditions in future enhancements.

MedGuardian is an AI-based medical prediction system designed to detect early signs of diseases using Machine Learning models.It analyzes health data such as age, blood pressure, sugar levels, symptoms, lifestyle factors, etc., and predicts the possibility of conditions like:

- Diabetes
- Heart disease
- Liver disease
- Other risk categories (extendable)

**The system includes:**

- **A clean and user-friendly interface**
  Allows patients or healthcare providers to input data easily.
- **A backend machine learning engine**
  Trained using real-world medical datasets to perform accurate predictions.
- **Automated data preprocessing**
  Ensures that the input data is cleaned, normalized, and ready for prediction.
- **Real-time prediction generation**
  Provides immediate feedback and risk scores based on model analysis.
- **Optional database connectivity**
  Stores patient reports, predictions, history, and logs using MongoDB.

It provides quick, reliable, and data-driven medical insights to assist doctors and patients in taking timely action.

## 3.1 Scope of the Work

**In-Scope**

- **Machine Learning–based disease prediction:**
  The system uses ML algorithms to predict disease risks based on input parameters.
- **Data preprocessing and model training:**
  Data is cleaned, normalized, encoded, and used for training logistic regression, SVM, or Random Forest models.
- **Interactive web interface:**
  Users can enter medical details, view predictions, and download reports.
- **Prediction results and risk score generation:**
  The system generates detailed probability scores and interpretable output.

- **Secure data handling:**
  Ensures privacy and proper storage of user information and reports.

- **Database storage (optional):**
  Enables saving user reports, medical records, and logs for future reference.

**Out-of-Scope**

- **Medical imaging analysis:**
  Predictions based on X-rays, MRI, or CT scans are not included.

- **Emergency diagnosis:**
  The system cannot replace real-time medical decision-making in emergencies.

- **Hospital equipment integration:**
  No direct connectivity with medical hardware or diagnostic equipment.

- **IoT monitoring devices:**
  Wearables or real-time physiological sensor integration is not supported.

## 3.2 Project Modules

### 1. User Input Module

- Provides forms for entering health data: age, glucose, blood pressure, symptoms, and lifestyle details.
- Performs validation to ensure accurate data entry (e.g., numeric ranges, mandatory fields).
- Handles incorrect or missing values gracefully.

### 2. Data Preprocessing Module

- Cleans raw medical data to remove noise and inconsistencies.
- Handles missing values using imputation techniques.
- Normalizes and encodes data for compatibility with ML algorithms.
- Ensures that the model receives high-quality inputs.

### 3. Machine Learning Prediction Module

- Loads the pre-trained ML model (Random Forest, Logistic Regression, SVM).
- Performs prediction based on the processed input data.
- Generates disease likelihood percentage and risk category.

- Ensures consistent and reliable output for every prediction.

**4. Result Visualization Module**

- Displays outputs in an understandable form:
  - Positive/Negative prediction
  - High/Low risk score
  - Probability percentage
- Generates a downloadable PDF/summary for patient or doctor use.
- Enhances clarity through charts or color indicators (optional).

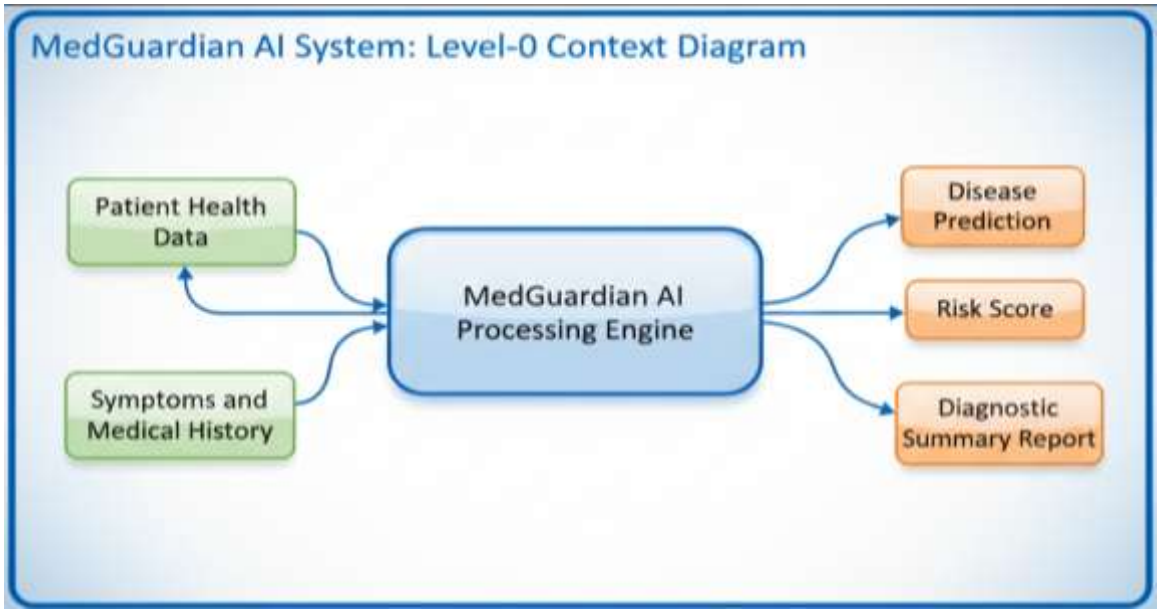
**5. Database Module (Optional)**

- Stores patient records, prediction histories, and reports.
- Uses MongoDB to store structured data such as user information and reports.
- Supports searching, filtering, and retrieving previous predictions.
- Maintains logs of user activities and interactions.

**6. Admin/Doctor Dashboard**

- Allows authorized users to review patient history.
- Tracks prediction accuracy, frequency, and system usage statistics.
- Provides administrative functionalities for user management (optional)

## 3.3 Context Diagram (High Level)



# 4 Implementation Methodology

1. **Requirement Analysis**
   - Identifying target users (patients, doctors).
   - Analyzing functional needs such as prediction accuracy, UI design, and security.
   - Defining non-functional requirements such as performance and scalability.

2. **Dataset Collection**
   - Medical datasets collected from Kaggle and open-source healthcare repositories.
   - Multiple datasets analyzed for quality and suitability for ML tasks.

3. **Data Preprocessing**
   - Removing duplicates, handling missing values, and normalizing input features.
   - Encoding categorical variables for model compatibility.
   - Splitting data into training and testing sets.

4. **Model Training & Evaluation**
   - Training multiple ML models and selecting the best-performing model.
   - Evaluating model accuracy using:
     - Confusion Matrix
     - Accuracy Score
     - Precision, Recall, F1-Score

5. **Web Application Development**
   - **Frontend:** Streamlit interface designed for ease of use.
   - **Backend:** Python functions handle prediction logic and data flow.
   - Implementing input validation, visualization components, and report generation.

6. **Integration and Testing**
   - Unit testing
   - Functional testing
   - User acceptance testing

7. **Deployment & Documentation**
   - Application deployed on Streamlit Cloud or local server.
   - Complete documentation including user guides, test cases, and maintenance notes

# 5 Technologies to be used

## 5.1 Software Platform

1. **Frontend: Streamlit**
   - Provides an interactive, browser-based UI.
   - Simplifies creation of input forms and visual outputs.
2. **Backend: Python**
   - Implements ML logic, predictions, APIs, and data handling.
   - Highly flexible and widely used in ML projects.
3. **Learning Libraries:**
   - Scikit-learn for model building and evaluation.
   - Pandas for data manipulation.
   - NumPy for numerical computations.
4. **Database: MongoDB**
   - Stores patient reports, login details, and prediction histories.
5. **Version Control: GitHub**
   - Manages code versions, commits, and team collaboration.
   - Useful for deployment workflows.

## 5.2 Hardware Platform

1. **4 GB RAM or higher**
   Required for running ML models and web applications smoothly.
2. **Operating Systems:**
   a. Windows
   b. Linux
   c. macOS
   The system is cross-platform and runs on all common OS environments.
3. **Web Browser:**
   a. Google Chrome (recommended)
   Used to access the Streamlit-based user interface.

## 5.3 Code

```
# app.py
import streamlit as st
import numpy as np
import pandas as pd
import pickle, json, os, joblib, io, time
from streamlit_lottie import st_lottie
from report import generate_pdf_report
import plotly.express as px
from chatbot import doctor_chatbot
from datetime import datetime

# DB helpers (MongoDB-compatible)
from db import create_user, authenticate_user, insert_report, get_reports_for_user, insert_chat,
get_chats_for_user
# Optional DB delete helper — if not implemented in your db module the code falls back to
session-state removal
try:
   from db import delete_report as db_delete_report
except Exception:
   db_delete_report = None

# -------------------- CONFIG --------------------
st.set_page_config(page_title="MedGuardian", page_icon=" 🧬 ", layout="wide")

# ---------- Settings ----------
TIMEOUT_MINUTES = 15  # auto-logout after this many minutes of inactivity

# ---------- Helpers ----------
def safe_rerun():
   """
   Modern Streamlit-safe rerun:
   1) try st.experimental_rerun (older versions)
   2) else update st.query_params (supported stable API) to force reload
   """
   try:
      st.experimental_rerun()
      return
   except Exception:
      pass
```

```
    try:
        params = dict(st.query_params or {})
        params["_ts"] = [str(int(time.time()))]
        st.query_params = params
        return
    except Exception:
        st.stop()


def update_last_active():
    st.session_state["last_active"] = time.time()


def check_auto_logout():
    """Log out user if inactivity exceeded TIMEOUT_MINUTES."""
    if st.session_state.get("user") and st.session_state.get("last_active"):
        elapsed = time.time() - st.session_state["last_active"]
        if elapsed > TIMEOUT_MINUTES * 60:
            st.session_state.user = None
            st.session_state.chat_history = []
            st.warning(f"Session timed out after {TIMEOUT_MINUTES} minutes of inactivity. Please
login again.")
            safe_rerun()


# ---------- Load Lottie ----------
def load_lottie(path):
    with open(path, "r", encoding="utf-8") as f:
        return json.load(f)

welcome_anim = None
if os.path.exists("welcome.json"):
    try:
        welcome_anim = load_lottie("welcome.json")
    except Exception:
        welcome_anim = None

# ---------- Load Models (robust) ----------
def _is_probable_pickle(path):
    """Quick heuristic check: file must be non-empty and start with pickle magic (0x80) or joblib
header."""
```

```
    try:
        sz = os.path.getsize(path)
        if sz < 100:
            print(f"[MODEL LOAD] {path} size too small ({sz} bytes).")
            return False
        with open(path, "rb") as f:
            head = f.read(4)
            if head.startswith(b'\x80') or head.startswith(b'\x93') or head.startswith(b'\x01'):
                return True
            if head.startswith(b'<!DO') or head.startswith(b'{') or head.startswith(b'<htm') or
head.startswith(b'<?xm'):
                print(f"[MODEL LOAD] {path} looks like text/HTML/JSON rather than a binary pickle.")
                return False
            return True
    except Exception as e:
        print("[MODEL LOAD] file check failed:", e)
        return False

def _safe_load(path):
    """Attempt joblib then pickle. Return loaded object or raise."""
    if not os.path.exists(path):
        raise FileNotFoundError(path)
    if not _is_probable_pickle(path):
        raise ValueError(f"File appears invalid or not a binary pickle: {path}")

    try:
        m = joblib.load(path)
        print(f"[MODEL LOAD] joblib.load successful: {path}")
        return m
    except Exception as e_job:
        print(f"[MODEL LOAD] joblib.load failed for {path} -> {e_job}. Trying pickle.load...")

    try:
        with open(path, "rb") as f:
            m = pickle.load(f)
        print(f"[MODEL LOAD] pickle.load successful: {path}")
        return m
    except Exception as e_pickle:
        print(f"[MODEL LOAD] pickle.load failed for {path} -> {e_pickle}")
        raise RuntimeError(f"Failed to load model file {path}: joblib error: {e_job}; pickle error:
{e_pickle}")
```

```
def _validate_model(obj, name="model"):
    """Ensure object has minimal methods used by your app (predict and optionally
predict_proba)."""
    if obj is None:
        return False
    if not hasattr(obj, "predict"):
        print(f"[MODEL LOAD] {name} does not implement .predict() — rejecting.")
        return False
    return True

@st.cache_resource
def load_models():
    models = {}
    paths = {
        "diabetes": "Diabetes/Diabetes_model.pkl",
        "heart": "Heart/heart_model.pkl",
        "kidney": "Kidney/kidney_model.pkl"
    }

    for key, p in paths.items():
        try:
            if os.path.exists(p):
                obj = _safe_load(p)
                if _validate_model(obj, name=key):
                    models[key] = obj
                else:
                    models[key] = None
                    st.warning(f"Loaded file for {key} but it's not a valid model (missing .predict). Using
fallback (disabled).")
            else:
                models[key] = None
                print(f"[MODEL LOAD] model file not found: {p}")
        except Exception as e:
            models[key] = None
            st.error(f"Failed to load {key} model from {p}: {e}")
            print(f"[MODEL LOAD] Exception while loading {p} ->", e)

    return models.get("diabetes"), models.get("heart"), models.get("kidney")

# Safe predict_proba wrapper: returns probability for positive class or None
def safe_predict_proba(m, X):
    try:
```

```
        if m is None:
            return None
        if hasattr(m, "predict_proba"):
            proba = m.predict_proba(X)
            if proba.ndim == 2 and proba.shape[1] >= 2:
                return proba[:,1]
            return proba[:,0]
        if hasattr(m, "decision_function"):
            df = m.decision_function(X)
            import math
            sigmoid = lambda x: 1/(1+math.exp(-x))
            return [sigmoid(v) for v in np.ravel(df)]
    except Exception as e:
        print("safe_predict_proba error:", e)
        return None

# Load models once (cached by Streamlit)
diabetes_model, heart_model, kidney_model = load_models()

# -------------------- Session init --------------------
if "user" not in st.session_state:
    st.session_state.user = None
if "reports" not in st.session_state:
    st.session_state.reports = []
if "auto_patient_id" not in st.session_state:
    st.session_state.auto_patient_id = f"MG-{datetime.now().strftime('%Y%m%d-%H%M%S')}"
if "last_active" not in st.session_state:
    st.session_state.last_active = None
if "chat_history" not in st.session_state:
    st.session_state.chat_history = []
if "welcome_done" not in st.session_state:
    st.session_state.welcome_done = False
if "start_time" not in st.session_state:
    st.session_state.start_time = time.time()

# -------------------- STYLES (small professional card style) --------------------
st.markdown("""
<style>
.card {background: linear-gradient(180deg, #ffffff, #f7fbff); padding: 22px; border-radius: 12px;
box-shadow: 0 6px 24px rgba(21,45,78,0.08);}
.auth-card {max-width:900px; margin: 20px auto;}
.center {display:flex; align-items:center; justify-content:center}
```

```
.small-muted {color: #6b7280; font-size:13px}
.delete-btn {background:#ff4b4b; color:white; padding:6px 10px; border-radius:8px}
.safe-btn {background:#7cd992; color:black; padding:6px 10px; border-radius:8px}
</style>
""", unsafe_allow_html=True)

# -------------------- AUTH (Full-page card) --------------------
def show_auth_page():
    """Full-width professional login/register card. Blocks access until login/register."""
    logo_img = "logo_splash.png" if os.path.exists("logo_splash.png") else "logo.png"

    st.markdown(f"""
    <div class='center auth-card card'>
      <div style='display:flex; gap:20px; align-items:center; width:100%'>
        <div style='flex:1'>
          <img src='{logo_img}' width='180' />
          <h2 style='margin-top:8px'>MedGuardian — Sign in to continue</h2>
          <p class='small-muted'>Secure access to your AI health reports and chat. Your data stays
private.</p>
        </div>
        <div style='flex:1.2'>
    """, unsafe_allow_html=True)

    auth_tab = st.radio("", ["Login","Register"], index=0, horizontal=True)

    if auth_tab == "Register":
        with st.form(key="register_form"):
            reg_user = st.text_input("Email / Username")
            reg_full = st.text_input("Full name")
            reg_phone = st.text_input("Phone (optional)")
            reg_pw = st.text_input("Password", type="password")
            create = st.form_submit_button("Create Account")
            if create:
                if not reg_user or not reg_pw:
                    st.error("Username and password required.")
                else:
                    ok = create_user(reg_user.strip(), reg_pw, reg_full.strip(), reg_phone.strip())
                    if ok:
                        st.success("Account created — please login.")
                    else:
                        st.error("Username already exists or error occurred.")
    else:
```

```
        with st.form(key="login_form"):
            login_user = st.text_input("Username or Email")
            login_pw = st.text_input("Password", type="password")
            login = st.form_submit_button("Login")
            if login:
                if not login_user or not login_pw:
                    st.error("Enter username and password.")
                else:
                    u = authenticate_user(login_user.strip(), login_pw)
                    if u:
                        st.session_state.user = u
                        update_last_active()
                        st.success(f"Welcome, {u.get('full_name') or u.get('username')}")
                        time.sleep(0.5)
                        safe_rerun()
                    else:
                        st.error("Invalid credentials")

    st.markdown("""
            </div>
        </div>
    </div>
    """, unsafe_allow_html=True)


# If not logged in — show auth page and block everything else
if not st.session_state.get("user"):
    show_auth_page()
    st.stop()

# -------------------- If logged in, check inactivity --------------------
check_auto_logout()

# -------------------- MAIN LAYOUT --- Sidebar with navigation (after auth) --------------------
with st.sidebar:
    sidebar_logo = "logo.png" if os.path.exists("logo.png") else ("logo_splash.png" if
os.path.exists("logo_splash.png") else None)
    if sidebar_logo:
        st.image(sidebar_logo, width=170)
    st.markdown("### 🔐 Account")
    st.markdown(f"**Signed in as:** {st.session_state.user['username']}")
```

```
    if st.button("Logout"):
        st.session_state.user = None
        st.session_state.chat_history = []
        st.success("Logged out.")
        safe_rerun()
    st.markdown("---")
    page = st.radio("Navigation", ["🏠 Dashboard","🩺 Health Scan","🤖 Doctor Chatbot"], index=0)
    st.info("Early Disease Prediction AI")

# -------------------- DASHBOARD (DB-backed when logged in) --------------------
if page == "🏠 Dashboard":
    update_last_active()
    st.markdown("<h2 style='text-align:center;color:#0066cc;'>📋 MedGuardian Analytics Dashboard</h2>", unsafe_allow_html=True)
    st.markdown("<p style='text-align:center;opacity:0.7;'>Patient Health Report Center</p>", unsafe_allow_html=True)
    st.markdown("<hr style='border:1px solid #cfe2ff;'>", unsafe_allow_html=True)

    # NOTE: user id returned by authenticate_user in MongoDB db.py is a string (ObjectId str)
    user_id = st.session_state.user.get('id')
    reports_db = []
    try:
        reports_db = get_reports_for_user(user_id)
    except Exception as e:
        st.error("Failed to fetch reports from DB: " + str(e))
        reports_db = []

    reports = []
    for r in reports_db:
        raw = r.get("raw") or {}
        if raw:
            # attach a DB id if present for deletion (db returns 'id' as string)
            raw['__db_id'] = r.get('id')
            reports.append(raw)
        else:
            entry = {
                "Patient ID": r.get("patient_id"),
                "Patient Name": r.get("patient_name"),
                "Phone": r.get("phone"),
                "Doctor Name": r.get("doctor_name"),
```

```
            "Referred By": r.get("referred_by"),
            "Sample Collected": r.get("sample_collected"),
            "Report Generated By": r.get("report_generated_by"),
            "Date": r.get("date"),
            "Condition": r.get("condition_name"),
            "Risk %": r.get("risk")
        }
        entry['__db_id'] = r.get('id')
        reports.append(entry)

# merge with session reports (local unsaved) — keep uniqueness
for r in st.session_state.reports:
    if not any((r.get('Patient ID') == rr.get('Patient ID') and r.get('Date') == rr.get('Date') and
r.get('Condition') == rr.get('Condition')) for rr in reports):
        r['__db_id'] = None
        reports.append(r)

st.session_state.reports = reports  # keep unified view

if not reports:
    st.info("📌 No reports found yet. Please run a prediction first.")
else:
    patients = sorted({r.get("Patient Name","Unknown") for r in reports})
    patient_filter = st.selectbox("👤 Filter by Patient", ["All"] + patients)
    disease_filter = st.selectbox("🩺 Filter by Condition", ["All","Heart","Diabetes","Kidney"])
    filtered = reports
    if patient_filter != "All":
        filtered = [r for r in filtered if r.get("Patient Name","Unknown") == patient_filter]
    if disease_filter != "All":
        filtered = [r for r in filtered if r.get("Condition") == disease_filter]
    if not filtered:
        st.warning("⚠ No records found for selected filters.")
    else:
        for i, r in enumerate(filtered):
            pid  = r.get("Patient ID","-")
            name = r.get("Patient Name","-")
            age = r.get("Age","-")
            gender = r.get("Gender","-")
            contact = r.get("Phone", r.get("Patient Contact","-"))
            referred_by = r.get("Referred By","-")
            condition = r.get("Condition","-")
```

```
            risk = r.get("Risk %", 0)
            bar_color = "#ff4b4b" if risk>=70 else "#ffb84d" if risk>=40 else "#7cd992"
            icon = " ❤️ " if condition=="Heart" else " 🍖 " if condition=="Diabetes" else " 🍼 "
            dbid = r.get('__db_id')

            with st.expander(f"{icon} {condition} — {name} | Risk: {risk}%", expanded=False):
              st.markdown(f"""
                <div style="background:rgba(255,255,255,0.60); padding:15px;border-radius:12px;
backdrop-filter:blur(8px); box-shadow: 0 6px 22px rgba(0,0,0,0.08); border-left:6px solid
{bar_color};">
                <b>👤 Patient Id:</b> {pid}<br>
                <b>👤 Patient:</b> {name}<br>
                <b>🎂 Age:</b> {age}     <b>| ⚧ Gender:</b> {gender} <br>
                <b>🔎 Contact:</b> {contact}     <b>| Referred By:</b>
{referred_by}<br><br>
                <b>🩺 Condition:</b> {condition}<br>
                <b>📊 Risk Level:</b> {risk}%<br>
              """, unsafe_allow_html=True)

              cols = st.columns([1,1,1])
              with cols[0]:
                try:
                  p = generate_pdf_report(condition, r, f"{condition} Report")
                  with open(p,"rb") as f:
                    st.download_button(f" 📄 Download {condition} Report", f,
file_name=f"{name}_{condition}_Report.pdf", key=f"pdf_{i}_{int(time.time())}")
                except Exception as e:
                  st.error("PDF generation error: "+str(e))
              with cols[1]:
                st.download_button(" 💾 Export (CSV)",
pd.DataFrame([r]).to_csv(index=False).encode('utf-8'),
file_name=f"{name}_{condition}_report.csv", key=f"csv_{i}_{int(time.time())}")
              with cols[2]:
                stable_key = f"del_btn_{dbid if dbid is not None else 'local'}_{i}"
                if st.button(" 🗑️ Delete Record", key=stable_key):
                  st.session_state['delete_candidate'] = {
                    'db_id': dbid,
                    'pid': pid,
                    'condition': condition,
                    'date': r.get('Date'),
                    'index': i
```

```
                    }

                cand = st.session_state.get('delete_candidate')
                if cand and cand.get('pid') == pid and cand.get('condition') == condition and
cand.get('date') == r.get('Date'):
                    st.warning("You are about to delete this record. This action cannot be undone.")
                    c1, c2 = st.columns([1,1])
                    if c1.button("Confirm Delete", key=f"confirm_del_{dbid if dbid is not None else
'local'}_{i}"):
                        removed_db = False
                        # db_delete_report expects string id for Mongo version (ObjectId string)
                        if cand.get('db_id') and db_delete_report is not None:
                            try:
                                ok = db_delete_report(cand.get('db_id'))
                                if ok:
                                    st.success("✔ Database record deleted successfully.")
                                    removed_db = True
                                else:
                                    st.warning("⚠ Database reported 0 rows affected (no deletion).")
                            except Exception as e:
                                st.error(f"❌ DB delete failed: {e}")
                                print("DB delete exception:", e)
                        else:
                            st.info("No DB delete helper available or record is unsaved (local-only).")

                        # Remove from local session view (always attempt)
                        try:
                            before = len(st.session_state.get('reports', []))
                            st.session_state.reports = [
                                rr for rr in st.session_state.get('reports', [])
                                if not (rr.get('Patient ID') == cand.get('pid') and rr.get('Condition') ==
cand.get('condition') and rr.get('Date') == cand.get('date'))
                            ]
                            after = len(st.session_state.get('reports', []))
                            if after < before:
                                st.success("✔ Report removed from local session view.")
                            else:
                                st.warning("⚠ Report not found in local session list to remove.")
                        except Exception as e:
                            st.error(f"Failed to remove local record: {e}")
                            print("Local delete exception:", e)
```

```
                    st.session_state.pop('delete_candidate', None)
                    update_last_active()
                    safe_rerun()

                if c2.button("Cancel", key=f"cancel_del_{dbid if dbid is not None else 'local'}_{i}"):
                    st.session_state.pop('delete_candidate', None)
                    safe_rerun()

            st.markdown("</div>", unsafe_allow_html=True)

        st.write("---")
        st.subheader("📈 Patient Risk Comparison")
        df = pd.DataFrame(filtered)
        if 'Patient Name' not in df.columns:
            df['Patient Name'] = df.get('Patient Name', pd.Series(["Unknown"]*len(df)))
        df['Risk %'] = pd.to_numeric(df['Risk %'], errors='coerce').fillna(0)
        fig = px.bar(df, x='Patient Name', y='Risk %', color='Condition', text='Risk %',
template='plotly_white',
                     color_discrete_map={"Heart":"#ff4b4b","Diabetes":"#ffb84d","Kidney":"#7cd992"})
        fig.update_traces(textposition="outside", marker_line_width=0.8,
marker_line_color='rgba(0,0,0,0.12)')
        fig.update_layout(yaxis_title="Risk %", xaxis_title="Patient", margin=dict(t=40,b=30),
bargap=0.25)
        st.plotly_chart(fig, use_container_width=True)
        csv = df.to_csv(index=False).encode('utf-8')
        st.download_button("💾 Download All Filtered Reports (CSV)", csv, "All_Reports.csv",
key=f"csv_{int(time.time())}")

# -------------------- HEALTH SCAN --------------------
elif page=="🩺 Health Scan":
    update_last_active()
    st.markdown("## 🧬 Smart Health Prediction")
    disease = st.selectbox("Select Test",["Heart","Diabetes","Kidney"])

    st.write("### 🧻 Patient Details")
    colA, colB = st.columns(2)
    doctor_options = ["Dr. A. Sharma", "Dr. B. Verma", "Dr. C. Roy", "Dr. D. Singh", "Other"]
    referred_options = ["Self", "Family Doctor", "OPD", "Lab Referral", "Clinic", "Other"]

    with colA:
```

```
    patient_id = st.text_input("Patient ID / Report No.",
value=st.session_state.get("auto_patient_id"), disabled=True,
                    help="This ID is auto-generated and linked to the report. (MG-YYYYMMDD-
HHMMSS)")
    st.session_state.patient_id = patient_id
    input_name = st.text_input("Patient Name", st.session_state.get("patient_name",""),
                    help="Full name of the patient. Example: 'John Doe'")
    st.session_state.patient_name = input_name.strip()
    patient_contact = st.text_input("Patient Contact Number",
st.session_state.get("patient_contact","+91 "),
                        help="Phone number including country code, e.g. +91 98765xxxxx")
    st.session_state.patient_contact = patient_contact.strip()
    st.caption(" 🔒 Patient Name and Patient Contact are required — they are used for record-
keeping in the report.")

  with colB:
    doctor_name = st.selectbox("Doctor Name", options=doctor_options, index=0,
                    help="Choose the reporting doctor's name. Select 'Other' to enter a custom
name.")
    if doctor_name == "Other":
      doctor_name_custom = st.text_input("Enter Doctor Name (Other)",
st.session_state.get("doctor_name",""),
                            help="If your doctor is not in the list, type the name here.")
      doctor_name = doctor_name_custom.strip() if doctor_name_custom.strip()!="" else
"Other"
    st.session_state.doctor_name = doctor_name

    referred_by = st.selectbox("Referred By", options=referred_options, index=0,
                    help="Who referred the patient — Self/Family/OPD/Lab etc. Select 'Other' to
specify.")
    if referred_by == "Other":
      referred_custom = st.text_input("Referred By (Other)",
st.session_state.get("referred_by",""),
                            help="If 'Other' was chosen, specify who referred the patient here.")
      referred_by = referred_custom.strip() if referred_custom.strip()!="" else "Other"
    st.session_state.referred_by = referred_by.strip()

    now_dt = datetime.now()
    st.markdown(f"**Sample Date:** {now_dt.strftime('%d-%m-%Y')}")
    st.session_state.sample_date_for_db = now_dt.strftime("%Y-%m-%d")
    st.session_state.sample_time_for_db = now_dt.strftime("%H:%M:%S")
```

```
        st.session_state.sample_datetime_for_db = now_dt.strftime("%Y-%m-%d %H:%M:%S")
        st.session_state.sample_collected = now_dt.strftime("%d-%m-%Y")

    report_generated_by = st.text_input("Report Generated By",
st.session_state.get("report_generated_by","MedGuardian AI Lab System"),
                                        help="Name of the system or person that generated this report.")
    st.session_state.report_generated_by = report_generated_by.strip()

    def validate_patient_details():
        missing = []
        if not st.session_state.get("patient_name"):
            missing.append("Patient Name")
        if not st.session_state.get("patient_contact"):
            missing.append("Patient Contact")
        if not st.session_state.get("doctor_name"):
            missing.append("Doctor Name")
        if not st.session_state.get("referred_by"):
            missing.append("Referred By")
        if missing:
            st.error("Please fill these mandatory patient details: " + ", ".join(missing))
            return False
        return True

    # ---------- HEART ----------
    if disease=="Heart":
        st.subheader("❤️ Heart Disease Input Panel")
        st.info("⚠️ If you don't know any value, leave it as default (Normal)")
        col1,col2 = st.columns(2)
        age = col1.number_input("Age",20,80,40, help="Patient's age in years. Range: 20-80")
        sex = col2.radio("Gender",["Male","Female"], help="Patient gender — Male or Female")
        sex_val = 1 if sex=="Male" else 0
        cp = st.select_slider("Chest Pain (0-3)", [0,1,2,3], help="Type of chest pain")
        trestbps = col1.number_input("Resting Blood Pressure (mm Hg)",90,200,120)
        chol = st.number_input("Cholesterol (mg/dL)",100,400,200)
        fbs = st.radio("Fasting Blood Sugar > 120 mg/dL",["No","Yes"])
        fbs_val = 1 if fbs=="Yes" else 0
        restecg = st.select_slider("Resting ECG (0-2)", [0,1,2])
        thalach = st.number_input("Max Heart Rate Achieved",80,200,140)
        exang = st.radio("Exercise Induced Angina",["No","Yes"])
        exang_val = 1 if exang=="Yes" else 0
        oldpeak = st.number_input("ST Depression (oldpeak)",0.0,6.0,1.0,0.1)
```

```
        slope = st.select_slider("Slope of ST (0-2)", [0,1,2])
        ca = st.select_slider("Number of Major Vessels Colored (0-4)", [0,1,2,3,4])
        thal = st.select_slider("Thalassemia (0-3)", [0,1,2,3])

        if st.button("🔍 Predict Heart Risk"):
            if not validate_patient_details():
                st.stop()
            data =
np.array([[age,sex_val,cp,trestbps,chol,fbs_val,restecg,thalach,exang_val,oldpeak,slope,ca,thal]])
            res = heart_model.predict(data)[0] if heart_model is not None else 0
            prob_arr = safe_predict_proba(heart_model, data)
            prob = float(prob_arr[0]*100) if prob_arr is not None else 0.0
            result = "⚠️ High Risk" if res==1 else "✅ Safe"
            st.success(f"{result} | Probability: {prob:.1f}%")
            report = {
                "Patient ID": st.session_state.get("patient_id","-"),
                "Patient Name": st.session_state.get("patient_name","-"),
                "Phone": st.session_state.get("patient_contact","-"),
                "Doctor Name": st.session_state.get("doctor_name","-"),
                "Referred By": st.session_state.get("referred_by","-"),
                "Sample Collected": st.session_state.get("sample_collected",
datetime.now().strftime("%d-%m-%Y %I:%M %p")),
                "Report Generated By": st.session_state.get("report_generated_by","MedGuardian AI
Lab System"),
                "Date": datetime.now().strftime("%d-%m-%Y %I:%M %p"),
                "Age": age,
                "Gender": "Male" if sex_val==1 else "Female",
                "Condition":"Heart",
                "Risk %": round(prob,2),
                "restbps": trestbps,
                "cp": cp,
                "Cholesterol": chol,
                "Fasting Blood Sugar": fbs_val,
                "Resting ECG": restecg,
                "Max Heart Rate": thalach,
                "Exercise Induced Angina": exang_val,
                "ST Depression": oldpeak,
                "ST Slope": slope,
                "Major Vessels Colored": ca,
                "Thalassemia": thal
            }
```

```
        key_tuple = (report["Patient ID"], report["Condition"], report["Date"])
        existing_keys = {(r.get("Patient ID"), r.get("Condition"), r.get("Date")) for r in
st.session_state.reports if r.get("Patient ID")}
        if key_tuple not in existing_keys:
          st.session_state.reports.append(report)
        # save to DB if logged in
        if st.session_state.get("user"):
          try:
            # pass user id (string) to insert_report — Mongo db.py handles it
            insert_report(st.session_state.user['id'], report)
            update_last_active()
            st.success("Report saved to your account.")
          except Exception as e:
            st.error("Failed to save report: " + str(e))
        else:
          st.info("Login to save this report to your account.")
        path = generate_pdf_report("Heart Disease", report, result)
        with open(path,"rb") as f:
          st.download_button(" 📄 Download Hospital Report",f,"Heart_Report.pdf")

    # ---------- DIABETES ----------
    if disease=="Diabetes":
      st.subheader(" 🧁 Diabetes Disease Input Panel")
      st.info(" ⚠️ If you don't know any value, leave it as default (Normal)")
      gender = st.radio("Gender",["Male","Female"])
      gender_val = 1 if gender=="Male" else 0
      age = st.number_input("Age",10,100,30)
      bmi = st.number_input("BMI",10.0,60.0,24.5)
      glu = st.number_input("Glucose (mg/dL)",50,300,110)
      hba = st.number_input("HbA1c (%)",4.0,15.0,5.7)
      hyt = st.radio("Hypertension",["No","Yes"])
      hyt_val = 1 if hyt=="Yes" else 0
      if st.button(" 🔍 Predict Diabetes"):
        if not validate_patient_details():
          st.stop()
        data = np.array([[gender_val,age,bmi,glu,hba,hyt_val]])
        res = diabetes_model.predict(data)[0] if diabetes_model is not None else 0
        prob_arr = safe_predict_proba(diabetes_model, data)
        prob = float(prob_arr[0]*100) if prob_arr is not None else 0.0
        result = " ⚠️ Diabetes Risk" if res==1 else " ✅ Normal"
        st.success(f"{result} | {prob:.1f}%")
```

```
            report = {
                "Patient ID": st.session_state.get("patient_id","-"),
                "Patient Name": st.session_state.get("patient_name","-"),
                "Phone": st.session_state.get("patient_contact","-"),
                "Doctor Name": st.session_state.get("doctor_name","-"),
                "Referred By": st.session_state.get("referred_by","-"),
                "Sample Collected": st.session_state.get("sample_collected",
datetime.now().strftime("%d-%m-%Y %I:%M %p")),
                "Report Generated By": st.session_state.get("report_generated_by","MedGuardian AI
Lab System"),
                "Date": datetime.now().strftime("%d-%m-%Y %I:%M %p"),
                "Age": age,
                "Gender": "Male" if gender_val==1 else "Female",
                "Condition":"Diabetes",
                "Risk %": round(prob,2),
                "BMI": bmi,
                "Glucose": glu,
                "HbA1c": hba,
                "Hypertension": hyt_val
            }
            key_tuple = (report["Patient ID"], report["Condition"], report["Date"])
            existing_keys = {(r.get("Patient ID"), r.get("Condition"), r.get("Date")) for r in
st.session_state.reports if r.get("Patient ID")}
            if key_tuple not in existing_keys:
                st.session_state.reports.append(report)
            if st.session_state.get("user"):
                try:
                    insert_report(st.session_state.user['id'], report)
                    update_last_active()
                    st.success("Report saved to your account.")
                except Exception as e:
                    st.error("Failed to save report: " + str(e))
            else:
                st.info("Login to save this report to your account.")
            path = generate_pdf_report("Diabetes", report, result)
            with open(path,"rb") as f:
                st.download_button(" 📄 Download Hospital Report",f,"Diabetes_Report.pdf")

        # ---------- KIDNEY ----------
        if disease=="Kidney":
            st.subheader(" 🧪 Kidney Disease Input Panel")
```

```
    st.info("⚠️ If you don't know any value, leave it as default (Normal)")
    gender_choice = st.radio("Gender",["Male","Female"])
    gender_val = 1 if gender_choice=="Male" else 0
    col1, col2 = st.columns(2)
    with col1:
      age = col1.number_input("Age",1,100,40)
      bp = col1.number_input("Blood Pressure (mmHg)",60,200,120)
      sg = col1.selectbox("Specific Gravity", [1.005,1.010,1.015,1.020,1.025])
      albumin = col1.select_slider("Albumin (0-5)", [0,1,2,3,4,5])
      sugar = col1.select_slider("Sugar (0-5)", [0,1,2,3,4,5])
      rbc = col1.selectbox("Red Blood Cells",["Normal","Abnormal","Don't Know"])
      pus = col1.selectbox("Pus Cell",["Normal","Abnormal","Don't Know"])
      pc = col1.selectbox("Pus Cell Clumps",["No","Yes","Don't Know"])
      bac = col1.selectbox("Bacteria",["No","Yes","Don't Know"])
      appetite = col1.selectbox("Appetite",["Good","Poor","Don't Know"])
      edema = col1.selectbox("Pedal Edema",["No","Yes","Don't Know"])
      aanemia = col1.selectbox("Aanemia",["No","Yes","Don't Know"])
    with col2:
      bgr = col2.number_input("Blood Glucose Random",0,500,120)
      bu = col2.number_input("Blood Urea",0,250,40)
      sc = col2.number_input("Serum Creatinine",0.0,15.0,1.1)
      sodium = col2.number_input("Sodium",100,170,140)
      potassium = col2.number_input("Potassium",2.0,10.0,4.5)
      hb = col2.number_input("Haemoglobin",5.0,20.0,13.0)
      pcv = col2.number_input("Packed Cell Volume",10,60,40)
      wbc = col2.number_input("White Blood Cell Count",2000,20000,8000)
      rbc_count = col2.number_input("Red Blood Cell Count",2.0,8.0,4.9)
      hypertension = col2.selectbox("Hypertension",["No","Yes","Don't Know"])
      diabetes = col2.selectbox("Diabetes Mellitus",["No","Yes","Don't Know"])
      cad = col2.selectbox("Coronary Artery Disease",["No","Yes","Don't Know"])
    conv = lambda x: 1 if str(x).strip().lower() in ["yes","poor","abnormal","1","true"] else 0
    df = pd.DataFrame([{
      "age":age,"blood_pressure":bp,"specific_gravity":sg,"albumin":albumin,"sugar":sugar,

"red_blood_cells":conv(rbc),"pus_cell":conv(pus),"pus_cell_clumps":conv(pc),"bacteria":conv(bac
),

"blood_glucose_random":bgr,"blood_urea":bu,"serum_creatinine":sc,"sodium":sodium,"potassiu
m":potassium,
      "haemoglobin":hb,"packed_cell_volume":pcv,"white_blood_cell_count":wbc,
      "red_blood_cell_count":rbc_count,"hypertension":conv(hypertension),
```

```
            "diabetes_mellitus":conv(diabetes),"coronary_artery_disease":conv(cad),
            "appetite":conv(appetite),"peda_edema":conv(edema),
            "aanemia": conv(aanemia)
        }])
        st.caption("Each field has a help tooltip — hover or tap to read meaning and expected
ranges.")
        if st.button("🫧 Predict Kidney Disease"):
            if not validate_patient_details():
                st.stop()
            res = kidney_model.predict(df)[0] if kidney_model is not None else 0
            prob_arr = safe_predict_proba(kidney_model, df)
            prob = float(prob_arr[0]*100) if prob_arr is not None else 0.0
            result = "⚠️ CKD Risk Detected" if res==1 else "✅ Kidneys Healthy"
            st.success(f"{result} | Risk: {prob:.2f}%")
            report = {
                "Patient ID": st.session_state.get("patient_id","-"),
                "Patient Name": st.session_state.get("patient_name","-"),
                "Phone": st.session_state.get("patient_contact","-"),
                "Doctor Name": st.session_state.get("doctor_name","-"),
                "Referred By": st.session_state.get("referred_by","-"),
                "Sample Collected": st.session_state.get("sample_collected",
datetime.now().strftime("%d-%m-%Y %I:%M %p")),
                "Report Generated By": st.session_state.get("report_generated_by","MedGuardian AI
Lab System"),
                "Date": datetime.now().strftime("%d-%m-%Y %I:%M %p"),
                "Age": age,
                "Gender": "Male" if gender_val==1 else "Female",
                "Condition":"Kidney",
                "Risk %": round(prob,2),
                "Blood Pressure": bp,
                "Specific Gravity": sg,
                "Albumin": albumin,
                "Sugar": sugar,
                "red_blood_cells": conv(rbc),
                "Pus Cell": conv(pus),
                "Pus Cell Clumps": conv(pc),
                "Bacteria": conv(bac),
                "Blood Glucose Random": bgr,
                "Blood Urea": bu,
                "Serum Creatinine": sc,
                "Sodium": sodium,
```

```
            "Potassium": potassium,
            "Haemoglobin": hb,
            "Packed Blood Volume": pcv,
            "White Blood Cell Count": wbc,
            "Red Blood Cell Count": rbc_count,
            "Hypertension": conv(hypertension),
            "Diabetes Mellitus": conv(diabetes),
            "Coronary Artery Disease": conv(cad),
            "Appetite": conv(appetite),
            "Peda Edema": conv(edema),
            "anemia": conv(aanemia)
        }
        key_tuple = (report["Patient ID"], report["Condition"], report["Date"])
        existing_keys = {(r.get("Patient ID"), r.get("Condition"), r.get("Date")) for r in
st.session_state.reports if r.get("Patient ID")}
        if key_tuple not in existing_keys:
            st.session_state.reports.append(report)
        if st.session_state.get("user"):
            try:
                insert_report(st.session_state.user['id'], report)
                update_last_active()
                st.success("Report saved to your account.")
            except Exception as e:
                st.error("Failed to save report: " + str(e))
        else:
            st.info("Login to save this report to your account.")
        path = generate_pdf_report("Kidney Disease", report, result)
        with open(path,"rb") as f:
            st.download_button(" 📄 Download Hospital Report",f,"Kidney_Report.pdf")

# -------------------- DOCTOR CHATBOT (improved + debug) --------------------
elif page == " 🤖 Doctor Chatbot":
    update_last_active()
    st.set_page_config(page_title="MedGuardian Doctor AI", layout="wide")
    st.markdown("<h2 style='text-align:center;'>🩺 MedGuardian — AI Doctor Chatbot</h2>",
unsafe_allow_html=True)

    use_gemini = st.checkbox(
        "Use Gemini (cloud LLM) for detailed answers (may require API key)",
        value=True
    )
```

```
    st.caption("Tip: Turn off Gemini to use only fast local rule-based replies.")

    api_present = True
    try:
        key = st.secrets.get("GEMINI_API_KEY") if hasattr(st, "secrets") else None
        if not key:
            key = os.environ.get("GEMINI_API_KEY")
        if use_gemini and not key:
            st.warning("Gemini API key not found. Add GEMINI_API_KEY to .streamlit/secrets.toml")
            api_present = False
    except Exception:
        api_present = False

    if "chat_history" not in st.session_state:
        st.session_state.chat_history = []
        if st.session_state.get("user"):
            try:
                rows = get_chats_for_user(st.session_state.user['id'], limit=500)
                for r in rows:
                    label = "You" if r['role'] == 'user' else "Doctor"
                    st.session_state.chat_history.append((label, r['message']))
            except Exception:
                pass

    chat_container = st.container()
    with chat_container:
        for sender, message in st.session_state.chat_history:
            if sender == "You":
                st.markdown(
                    f"<div style='background:#D5F5E3;padding:12px;border-radius:12px;margin:8px;text-
align:right;'>"
                    f"<b>👤 You:</b> {message}</div>",
                    unsafe_allow_html=True)
            else:
                st.markdown(
                    f"<div style='background:#E8DAEF;padding:12px;border-radius:12px;margin:8px;text-
align:left;'>"
                    f"<b>🤖 Doctor AI:</b> {message}</div>",
                    unsafe_allow_html=True)

    st.write("---")
```

```
with st.form(key="chat_form", clear_on_submit=True):
    cols = st.columns([5, 1])
    with cols[0]:
        user_input = st.text_input(
            "Type your question...",
            placeholder="Ask about symptoms, diet, tests etc...",
            key="chat_input"
        )
    with cols[1]:
        send = st.form_submit_button("Send", use_container_width=True)

    if send and user_input.strip() != "":
        update_last_active()

        st.session_state.chat_history.append(("You", user_input))
        if st.session_state.get("user"):
            try:
                insert_chat(st.session_state.user['id'], "user", user_input)
            except Exception:
                pass

        th = st.empty()
        th.markdown(
            "<div style='background:#FFF8DC;padding:12px;border-radius:12px;margin:8px;'>"
            "<b>🤖 Doctor AI:</b> <i>Thinking...</i></div>",
            unsafe_allow_html=True)

        if use_gemini and not api_present:
            reply = "Gemini key missing — enable GEMINI_API_KEY to get full responses."
        else:
            try:
                reply = doctor_chatbot(
                    user_input,
                    user_id=st.session_state.user['id'] if st.session_state.get("user") else None,
                    use_gemini=use_gemini,
                    style="detailed",
                    max_tokens=512,
                    temperature=0.2,
                    top_p=0.95
                )
```

```
            except Exception as e:
                reply = f"Sorry — chatbot error occurred ({e})."
                print("\n--- Chatbot Error ---\n", e)

        th.empty()

        st.session_state.chat_history.append(("Doctor", reply))
        if st.session_state.get("user"):
            try:
                insert_chat(st.session_state.user['id'], "bot", reply)
            except Exception:
                pass

        safe_rerun()

if st.button(" 🧹 Clear Chat History"):
    st.session_state.chat_history = []
    safe_rerun()
```

## 5.4 Tools, if any

The development of the *MedGuardian: AI-Powered Early Disease Detection System* required the use of several software tools to support coding, testing, data handling, and deployment. The tools used in this project are listed below:

**1. Jupyter Notebook**

Jupyter Notebook was used for exploratory data analysis (EDA), dataset preprocessing, and initial machine learning model development. Its interactive environment allowed step-by-step execution, visualization of data patterns, and evaluation of model performance metrics such as accuracy, precision, and recall. It also enabled easy experimentation and debugging during early stages of model building.

**2. Visual Studio Code (VS Code)**

VS Code served as the primary code editor for writing Python scripts, organizing project files, implementing the backend logic, and integrating the Streamlit-based user interface. The editor's rich extensions, debugging tools, and version control integration made development efficient and well-structured.

**3. MongoDB Atlas**

MongoDB Atlas, a cloud-based NoSQL database service, was used to store user information, prediction reports, and chat logs securely. It provides a scalable and reliable storage solution, supports JSON-style documents, and allows quick retrieval of patient histories and system records. Its cloud interface also simplified database monitoring and management.

**4. Git & GitHub**

Git was used for version control to track changes throughout the development lifecycle. GitHub provided a cloud repository to store source code, manage commits, collaborate, and maintain backup versions of the project. It also served as a deployment reference for hosting and reviewing the project externally.

# 6 Advantages of this Project

The *MedGuardian* system provides several advantages that make it a valuable contribution to healthcare technology:

1. **Early Disease Detection**
   Identifies disease risks at an early stage, allowing timely intervention and reducing complications.
2. **Fast and Automated Predictions**
   The AI engine processes patient data instantly, offering risk evaluation within seconds.
3. **High Accuracy and Reliability**
   Machine Learning models trained on real-world datasets improve prediction accuracy and consistency compared to manual analysis.
4. **User-Friendly Interface**
   Even non-technical users can easily input data and understand prediction results.
5. **Low Cost and Scalable**
   Compared to traditional diagnostic systems, MedGuardian is cost-effective and can be scaled to support large numbers of users.
6. **Centralized Data Storage (Optional)**
   Medical reports and histories can be stored securely for future reference.
7. **Supports Doctors and Hospital Workflows**
   AI-based insights assist doctors in making informed decisions and reduce diagnostic workload.
8. **Customizable and Extensible**
   New disease models, datasets, and features can be easily added in future versions.

# 7 Assumptions, if any

The development of this project is based on the following assumptions:

- The datasets used for training are accurate and represent actual medical conditions.
- Users provide correct and complete input information for accurate predictions.
- The system is used as a decision-support tool, not as a replacement for medical diagnosis.

- A stable internet connection is available if the application is deployed online.
- MongoDB or any other selected database is properly configured during deployment.

If there are no additional assumptions required by your project guide, you may write:

# 8 Future Scope and further enhancement of the Project

MedGuardian has significant potential for expansion. The following future enhancements can improve its capabilities:

1. **Addition of More Disease Models**
   Integrating models for cancer risk, kidney disorder detection, anemia prediction, etc.
2. **Integration with Wearable Devices**
   Connecting with smartwatches and IoT devices for real-time health monitoring.
3. **Mobile App Development**
   Creating an Android/iOS version for easier access.
4. **Electronic Health Record (EHR) Integration**
   Directly accessing hospital systems to fetch patient history.
5. **Advanced ML and Deep Learning Algorithms**
   Implementing neural networks, CNN/LSTM models for improved accuracy.
6. **AI-Powered Chat Assistant**
   Guiding users with personalized recommendations based on their health risks.
7. **Automated Report Generation with Medical Insights**
   Adding interpretation texts, lifestyle suggestions, and warnings.
8. **Multilingual Support**
   Supporting regional languages like Hindi, Tamil, Telugu for wider adoption.
9. **Role-Based Access System**
   Separate dashboards for doctors, admins, and patients.

# 9 Project Repository Location

| S# | Project Artifacts (softcopy) | Location (Folder Name, Drive Link etc.) | Verified by Project Guide | Verified by HOD |
|---|---|---|---|---|
| 1. | Project Synopsis Report (Final Version) | https://github.com/OmChauhan1111/MedGuardian/Synopsis | Name and Signature | Name and Signature |
| 2. | Project Progress updates | https://github.com/OmChauhan1111/MedGuardian/Progress | Name and Signature | Name and Signature |
| 3. | Project Report (Final Version) | https://github.com/OmChauhan1111/MedGuardian/Report | Name and Signature | Name and Signature |
| 4. | Test Repository | https://github.com/OmChauhan1111/MedGuardian/Testing | Name and Signature | Name and Signature |
| 5. | Any other document, give details (certificate Copy) | https://github.com/OmChauhan1111/MedGuardian/Certificates | Name and Signature | Name and Signature |

# 10 Definitions, Acronyms, and Abbreviations

| Abbreviation | Description |
|---|---|
| **AI** | Artificial Intelligence |
| **ML** | Machine Learning |
| **UI** | User Interface |
| **UX** | User Experience |
| **API** | Application Programming Interface |
| **DB** | Database |
| **EDA** | Exploratory Data Analysis |
| **CSV** | Comma Separated Values (file format) |
| **EHR** | Electronic Health Record |
| **SVM** | Support Vector Machine (ML Algorithm) |
| **KNN** | K-Nearest Neighbors (ML Algorithm) |
| **MongoDB** | NoSQL database used for document storage |
| **Streamlit** | Web framework for data apps in Python |

| | |
|---|---|
| **Accuracy Score** | Metric used to evaluate ML model performance |
| **AI** | Artificial Intelligence |

# 11 Conclusion

The *MedGuardian: AI-Powered Early Disease Detection System* successfully demonstrates how modern Machine Learning techniques can be applied to real-world healthcare challenges. By analyzing patient health parameters and generating risk-based predictions, the system provides a fast, accurate, and reliable method for supporting early diagnosis. This not only reduces the burden on medical professionals but also empowers patients to make informed decisions about their health.

The project fulfilled all key objectives, including data preprocessing, model training, risk prediction, user-friendly interface design, and secure report storage using MongoDB. The integration of AI-driven insights and a simple web-based interface makes the system highly accessible for both patients and healthcare practitioners.

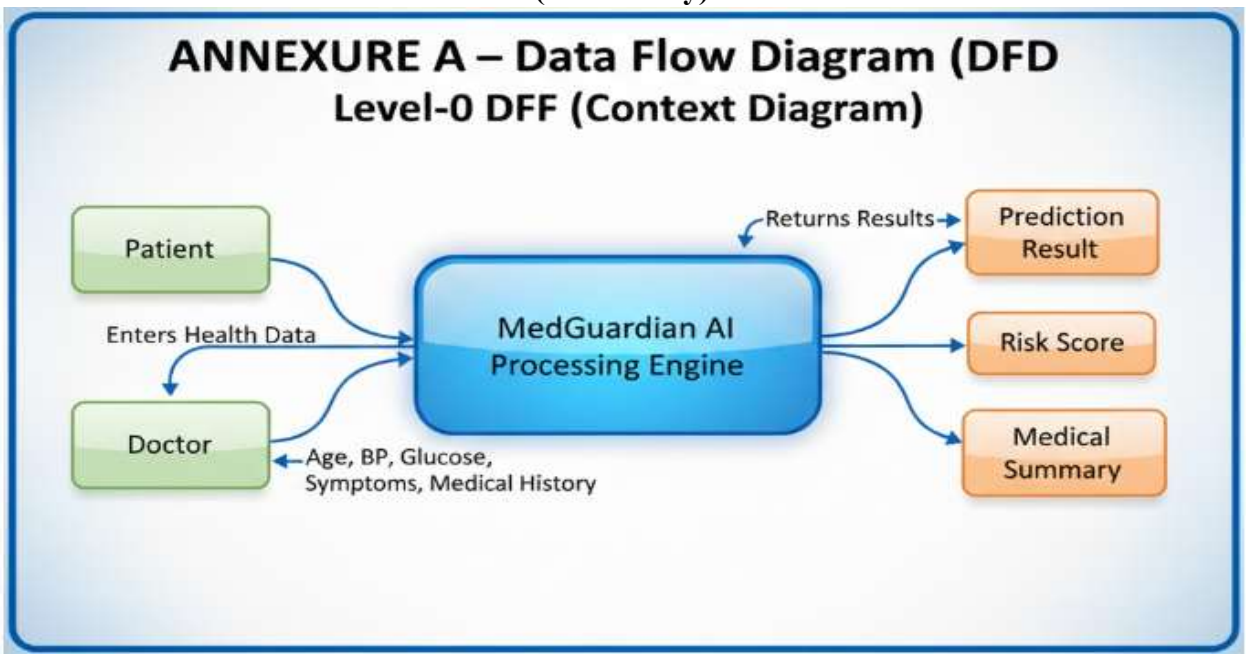
While the current version focuses on common diseases such as diabetes, heart disease, and liver-related conditions, there is significant scope to enhance the system by incorporating additional datasets, more advanced deep learning models, real-time monitoring capabilities, and integration with hospital management systems

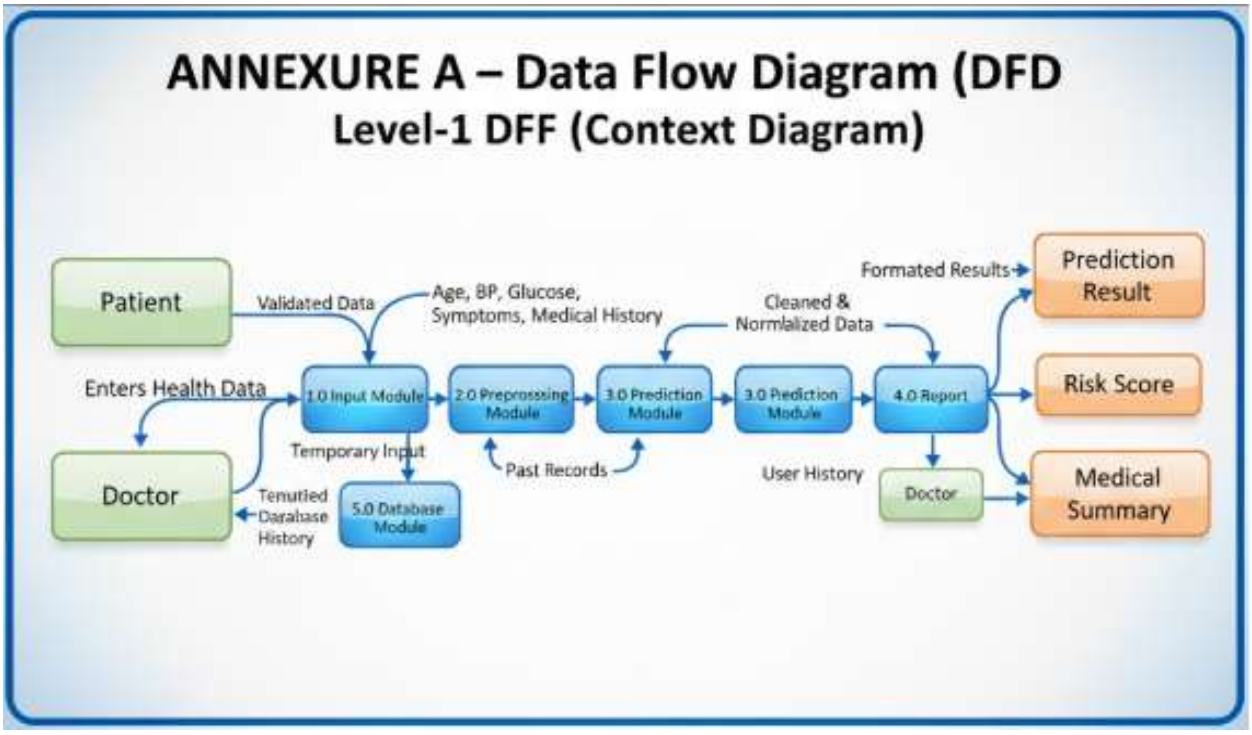
Overall, this project highlights the importance of AI in healthcare innovation and demonstrates how technology can contribute to early detection, improved treatment outcomes, and enhanced patient care.

# 12 References

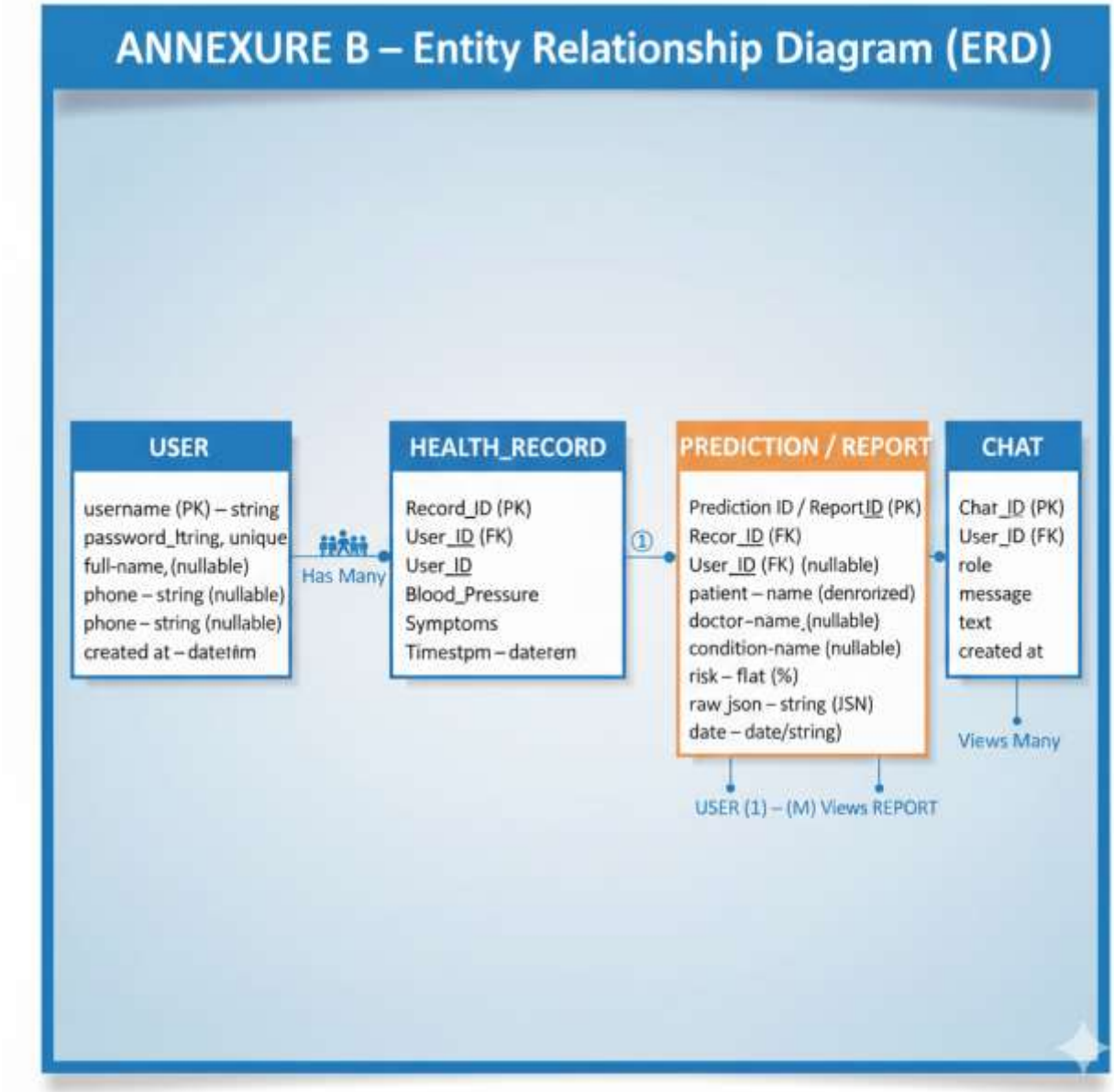

- Kaggle Medical Datasets – https://www.kaggle.com
- Scikit-learn Documentation – https://scikit-learn.org
- Streamlit Documentation – https://docs.streamlit.io
- MongoDB Atlas Documentation – https://www.mongodb.com/atlas
- Python Official Documentation – https://docs.python.org
- NumPy Documentation – https://numpy.org
- Pandas Documentation – https://pandas.pydata.org
- Healthcare AI Modeling Articles & Online Tutorials

# Annexure A
# Data Flow Diagram (DFD)
**(Mandatory)**



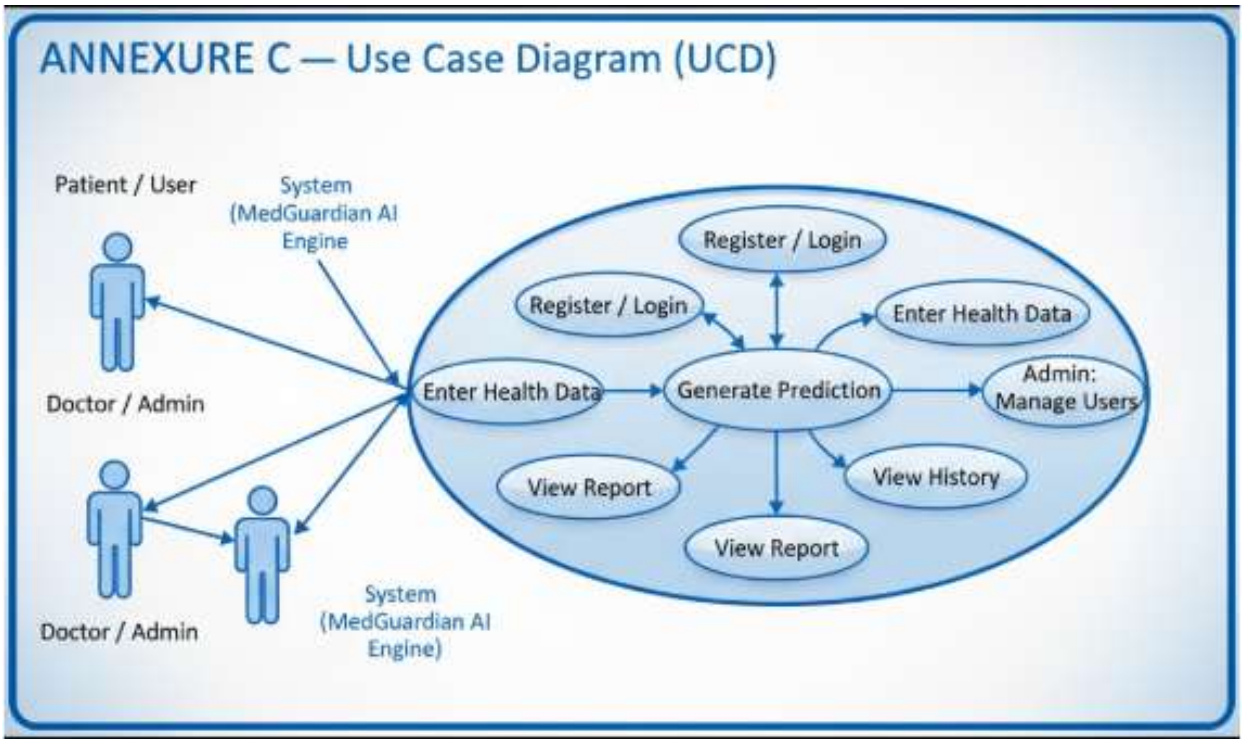



# Annexure B
# Entity-Relationship Diagram (ERD)
**(Mandatory)**



# Annexure C
## Use-Case Diagram (UCD)

**(Optional)**



# Annexure D
# Data Dictionary (DD)
**(Mandatory)**

## User Table (users)

| Fields | Data type | Description |
|---|---|---|
| _id | ObjectId | MongoDB internal document id |
| username | String | Unique username (stored lowercase) |
| password_hash | String | Bcrypt hashed password |
| full_name | String | Full name of the user |
| phone | String | Contact number |
| created_at | DateTime | UTC timestamp of account creation |

## Reports Table (reports)

| Fields | Data type | Description |
|---|---|---|
| _id | ObjectId | MongoDB document id |
| user_id | String | Stringified ObjectId referencing users._id |
| patient_id | String | External patient id (if any) |
| patient_name | String | Patient full name |
| phone | String | Patient phone |
| doctor_name | String | Doctor who reviewed or generated report |
| referred_by | String | Referrer source / department |
| sample_collected | String / Date | Sample collection info |
| report_generated_by | String | System or user who generated report |
| date | String / Date | Date of the report (domain date) |
| condition_name | String | Name of predicted condition |
| risk | Float | Risk percentage (0.0 - 100.0) |
| raw_json | String | Full report JSON (stringified) for audit |

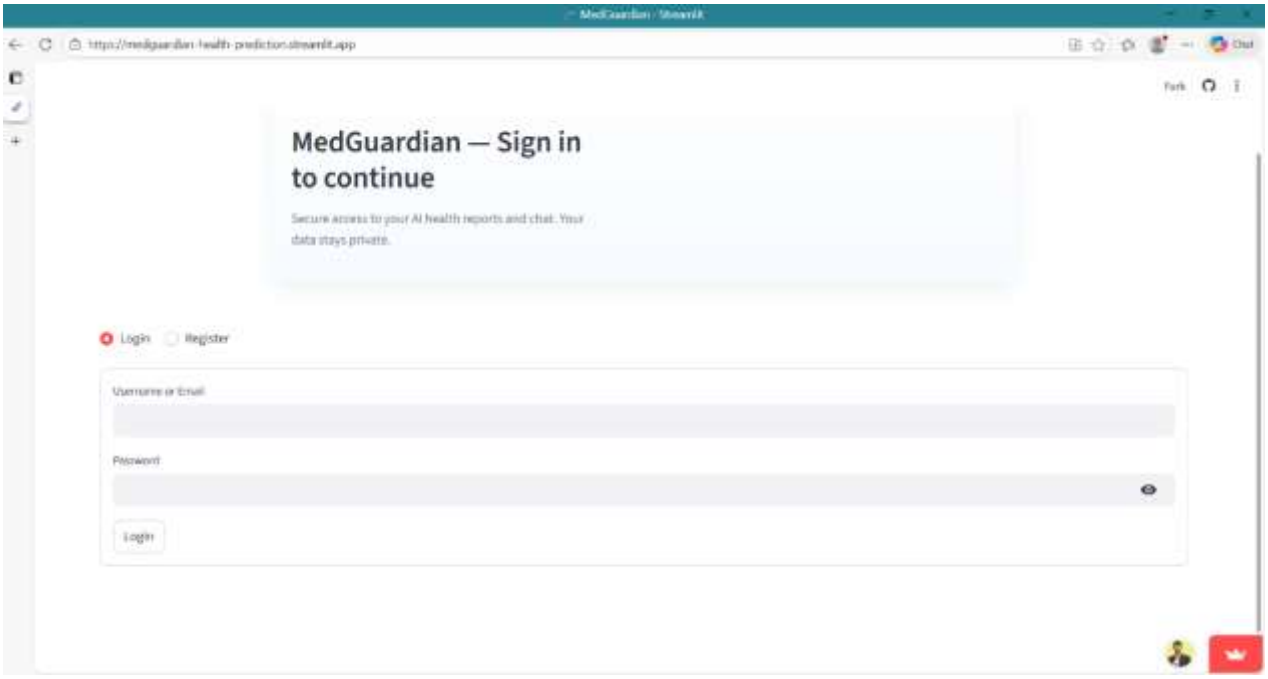
| | | |
|---|---|---|
| created_at | DateTime | Timestamp when document inserted |

## Chats Table (chats)

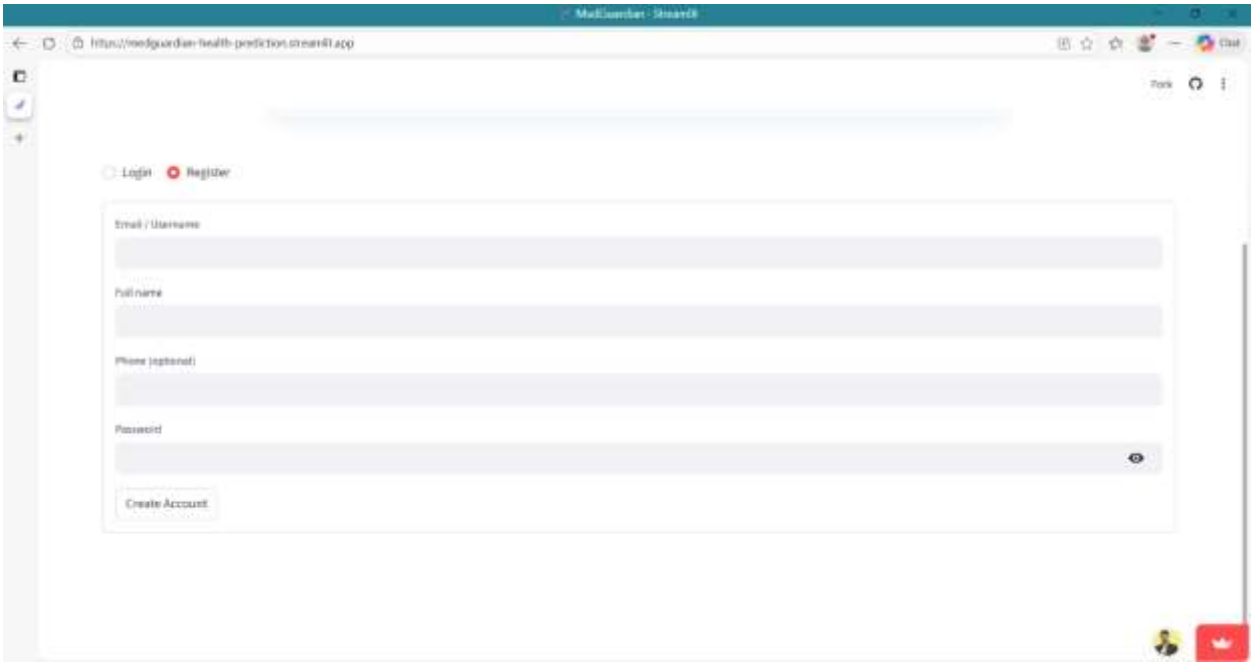
| Fields | Data type | Description |
|---|---|---|
| _id | ObjectId | MongoDB document id |
| user_id | String | Stringified user id referencing users._id |
| role | String | Role of message sender (user, assistant, doctor) |
| message | String | Message text or content |
| created_at | DateTime | Message timestamp (UTC) |

# Annexure E
# Screen Shots

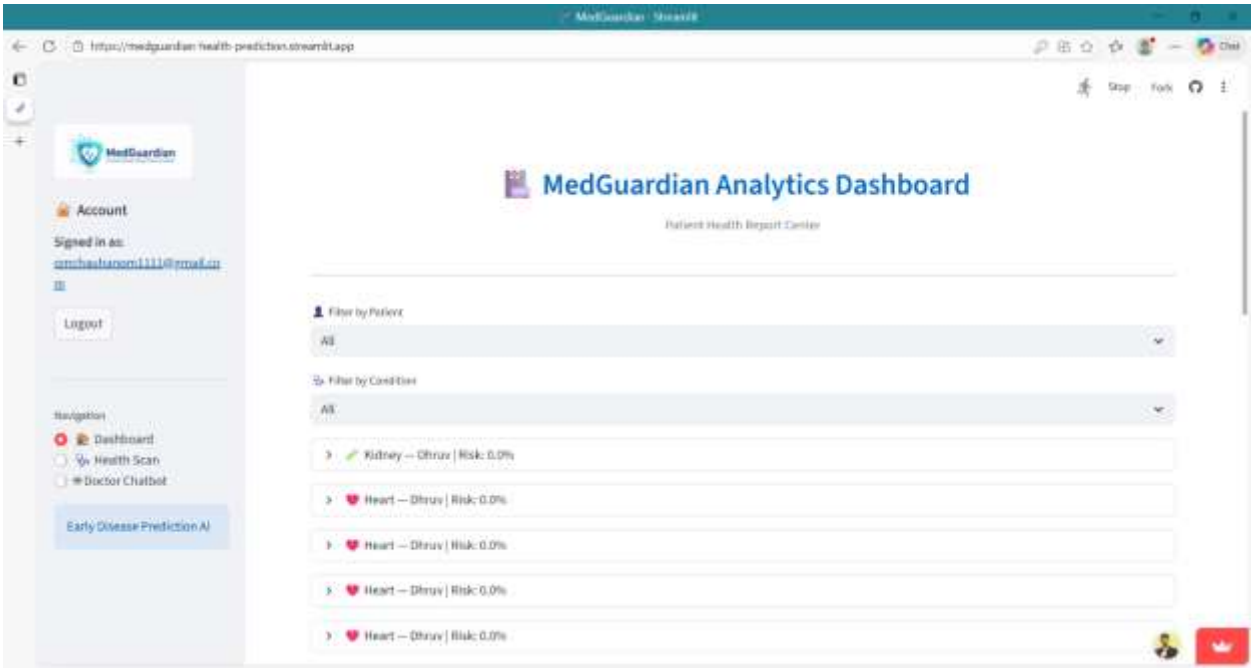
## Login Page:



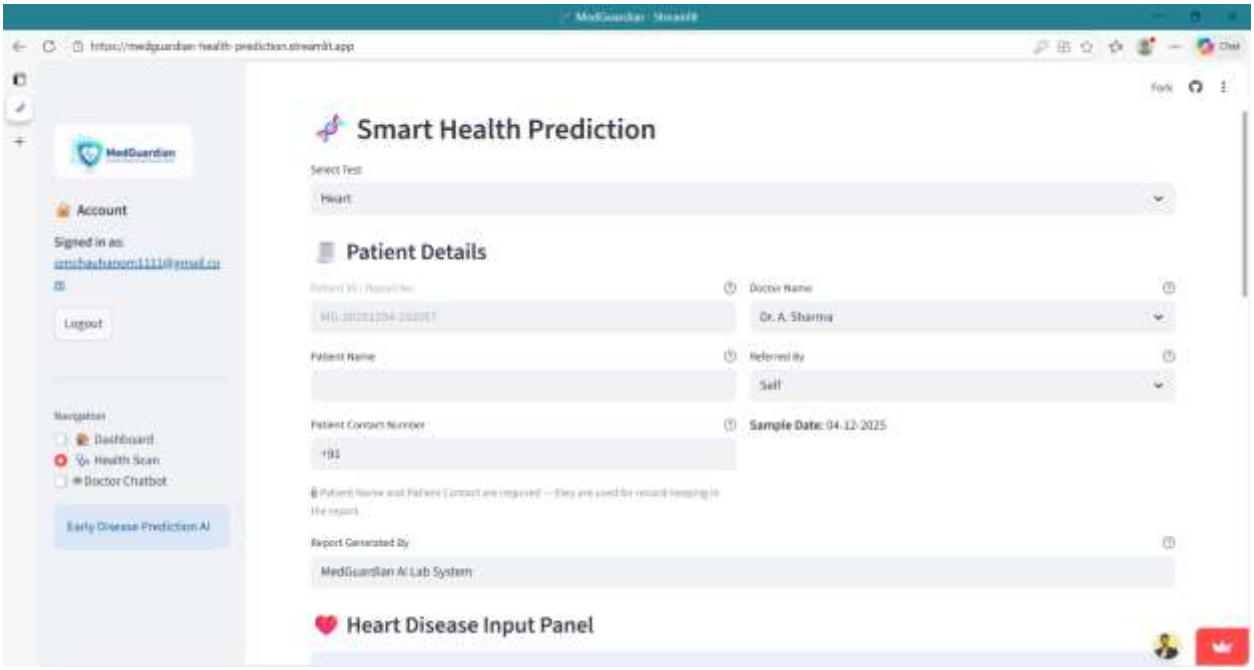

## Register Page:



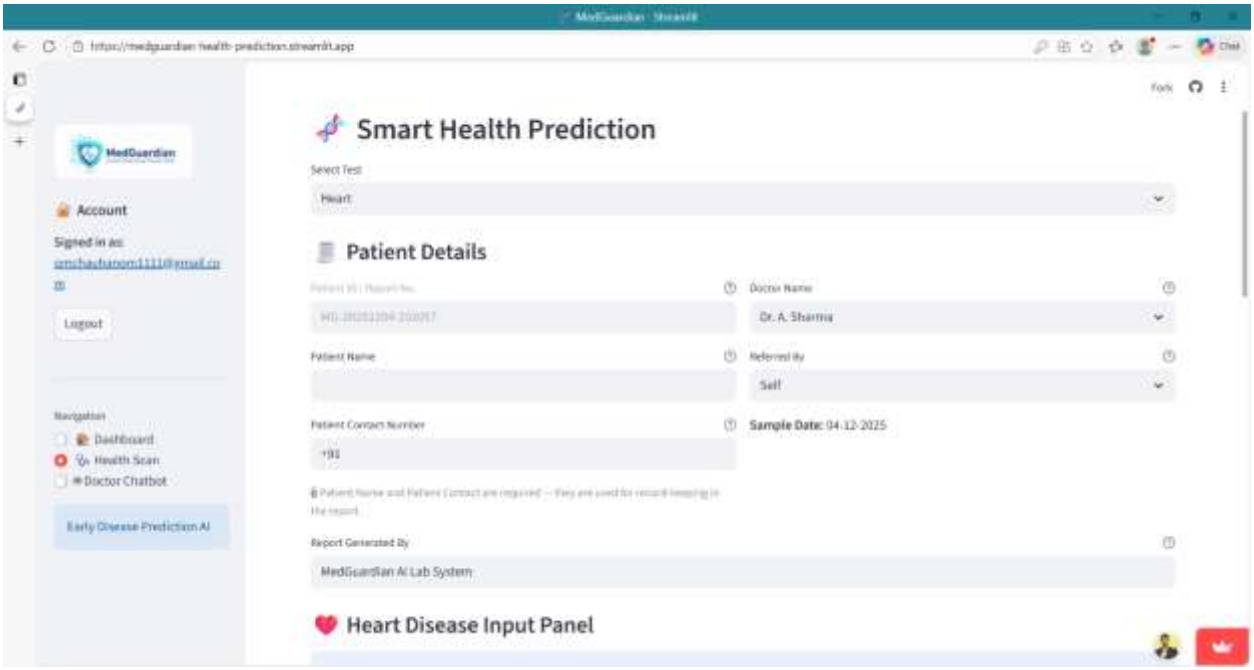
## Dashboard Page:

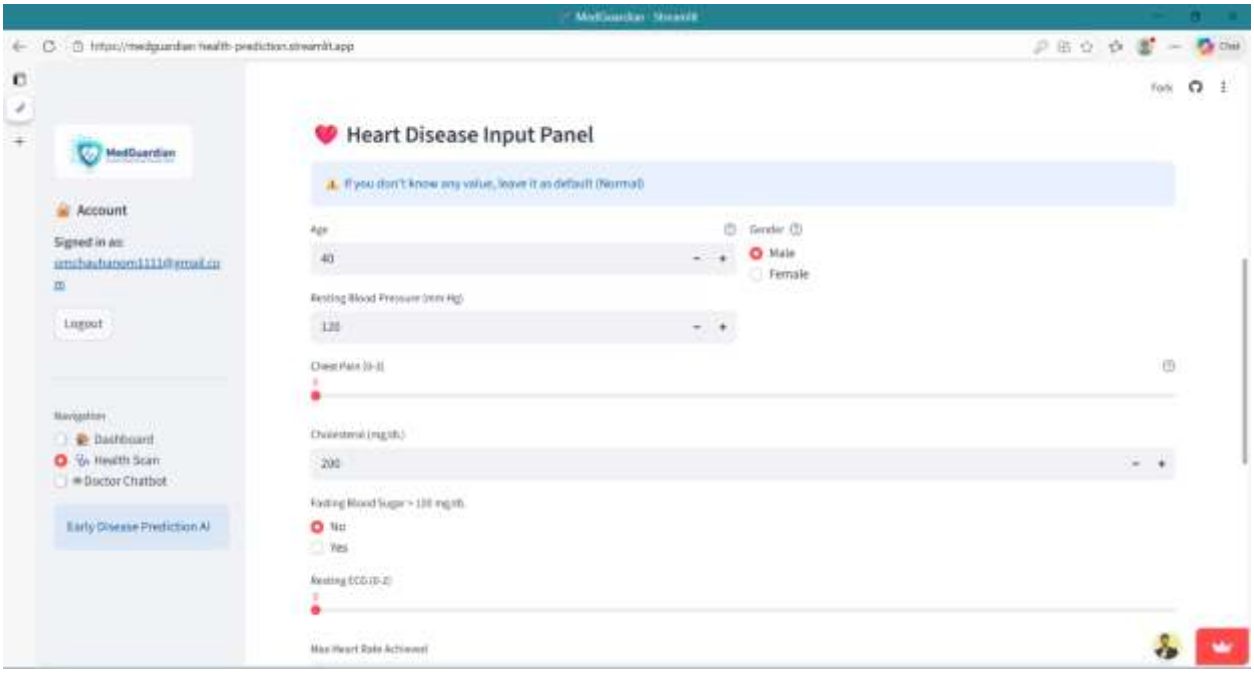


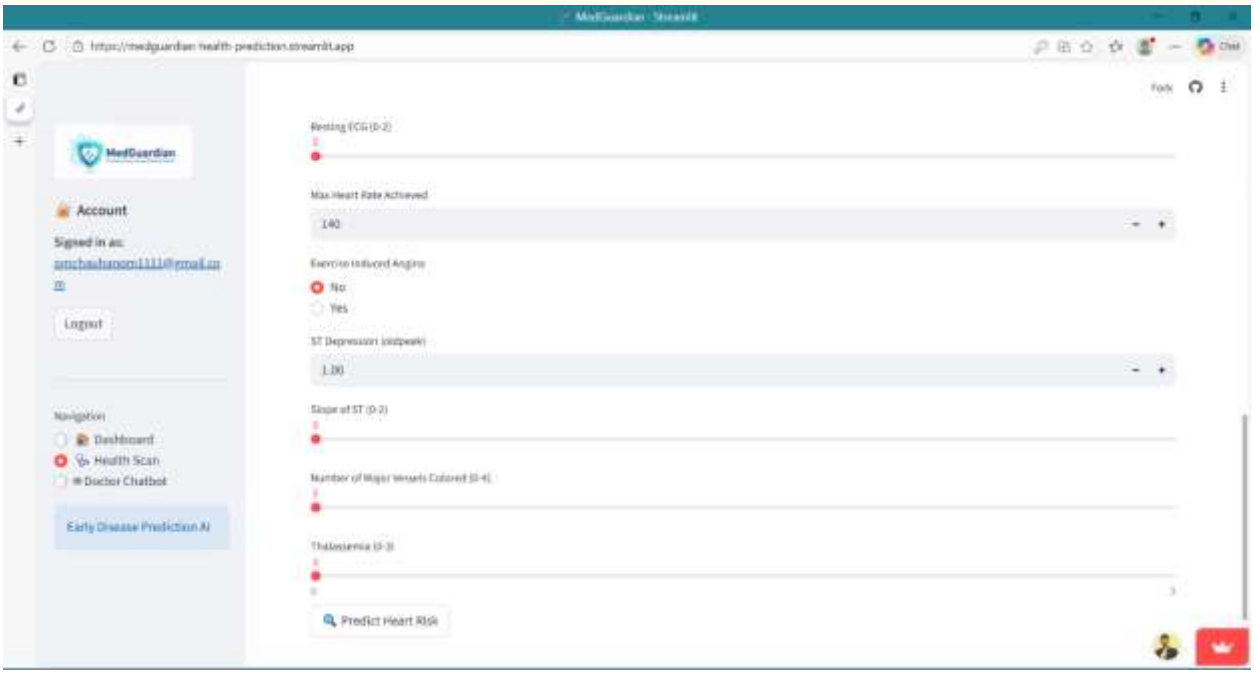

## Health Scan Page:



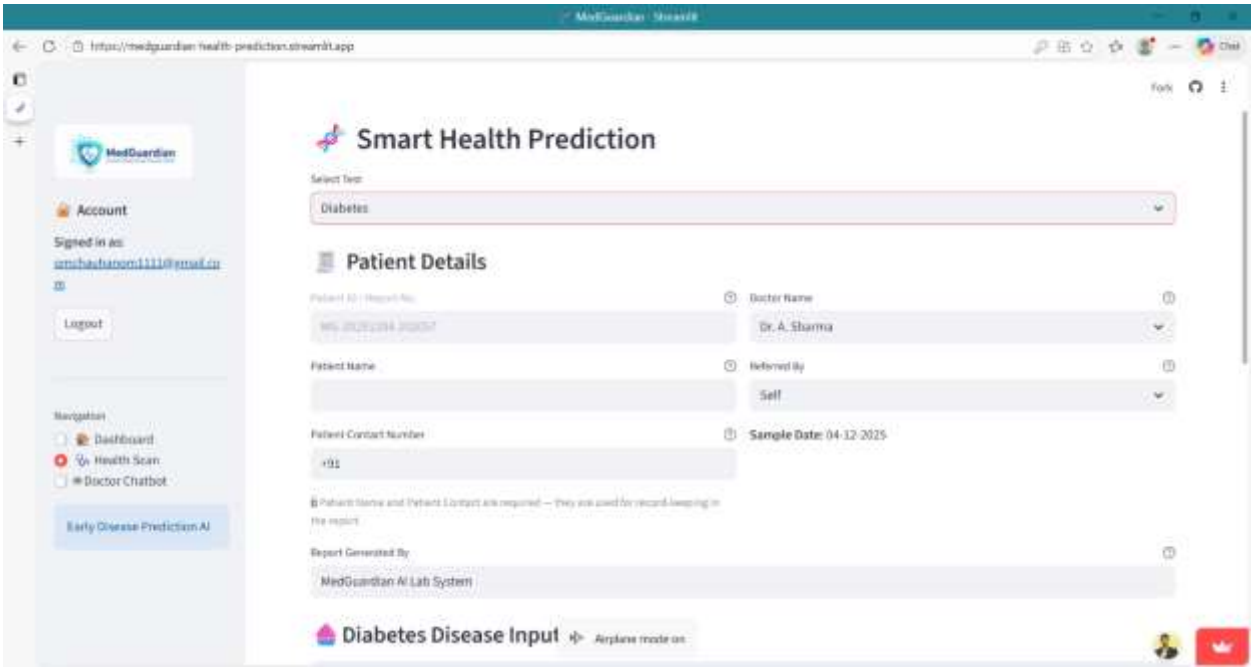
## Heart Prediction Page:

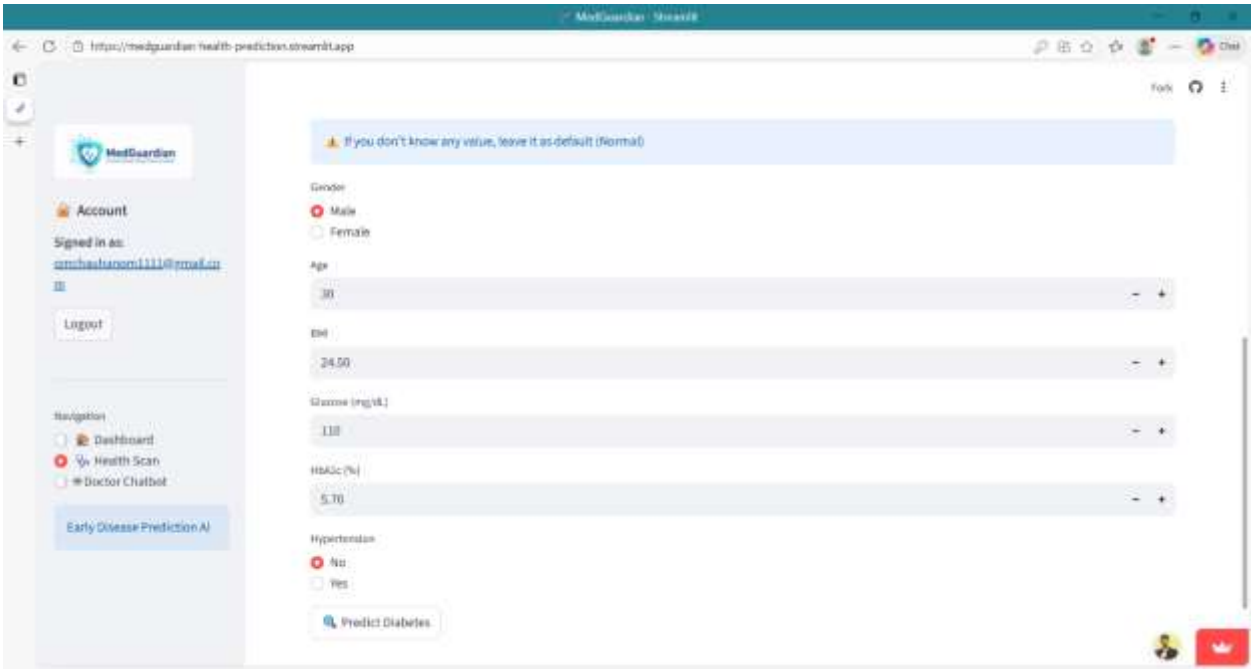


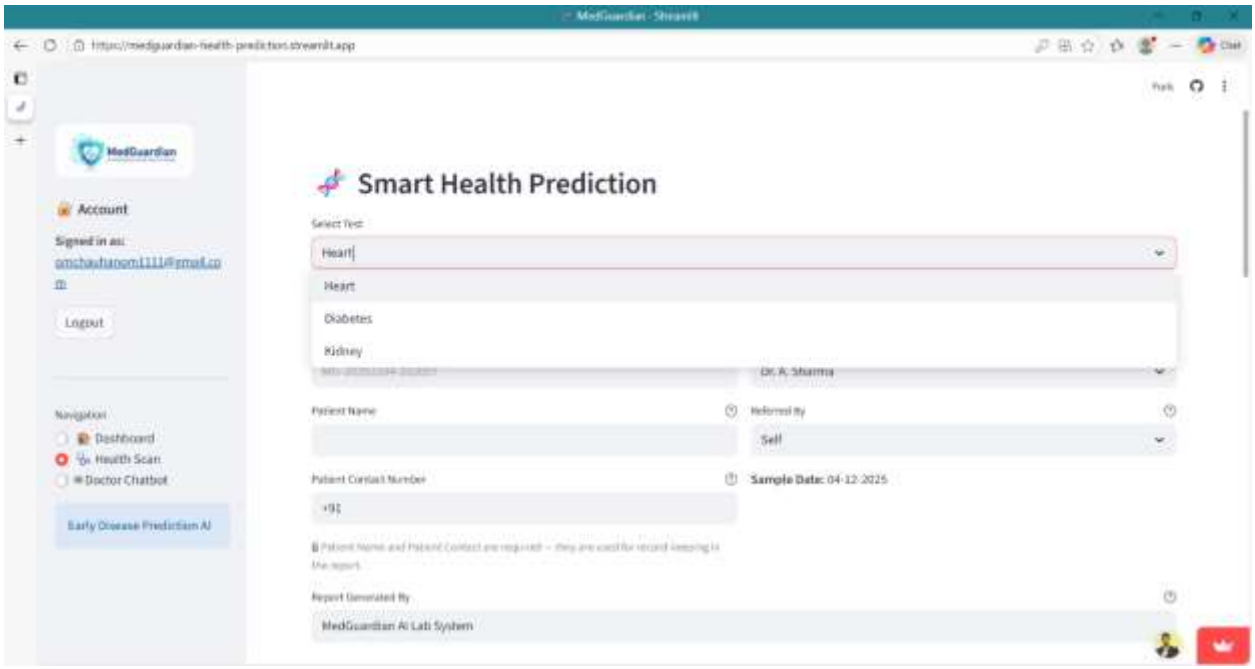


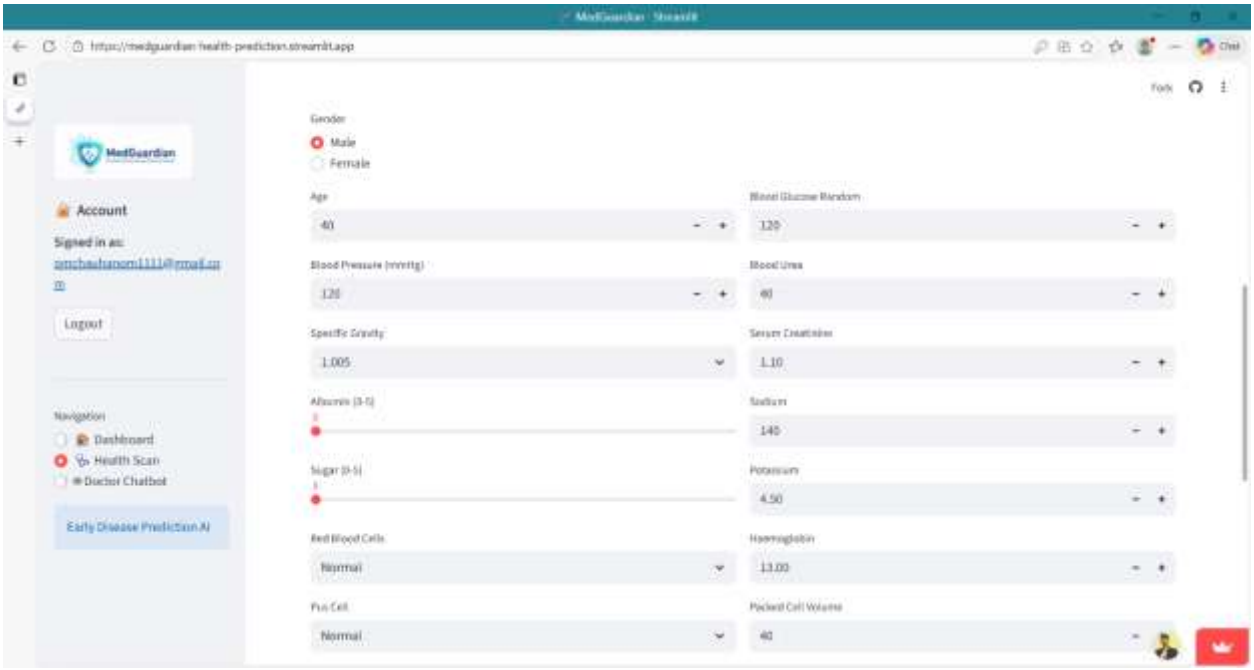

## Diabetes Prediction Page:

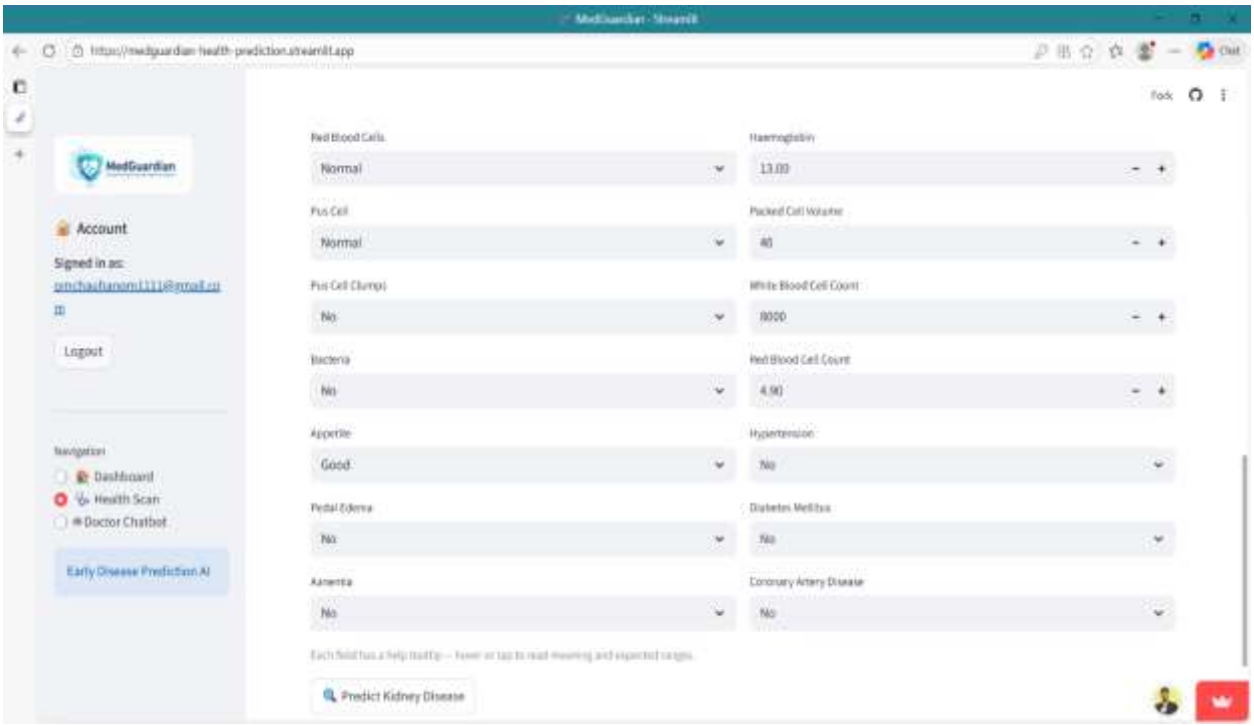




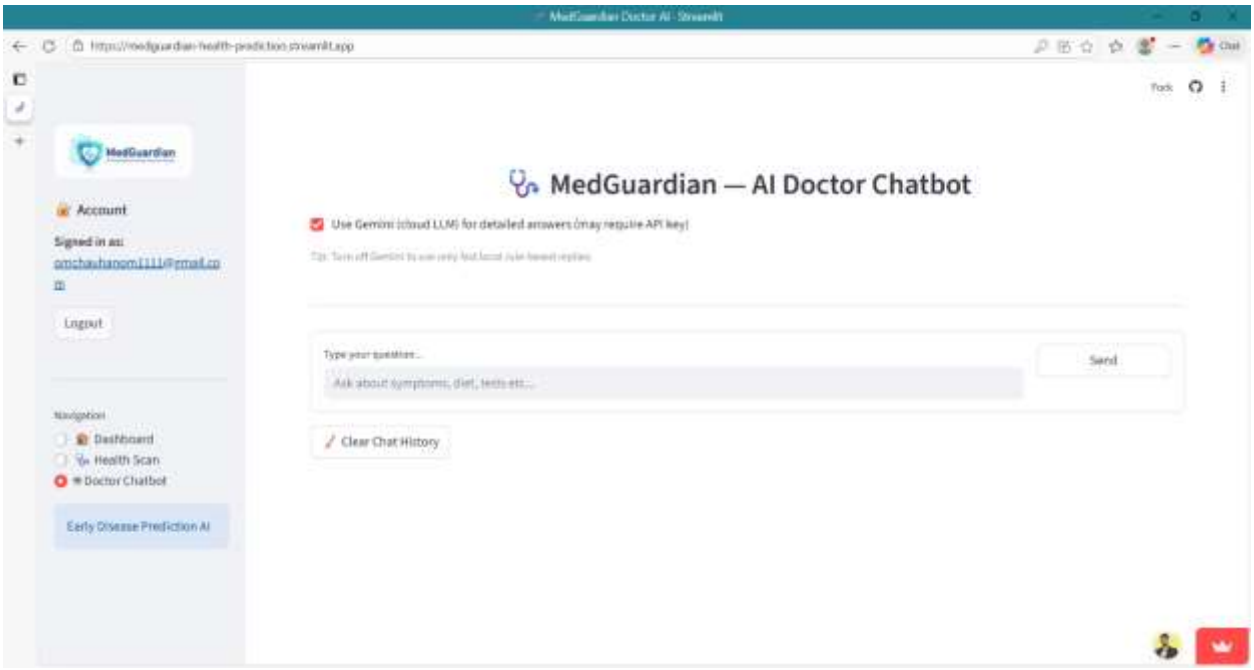

## Kidney Prediction Page:



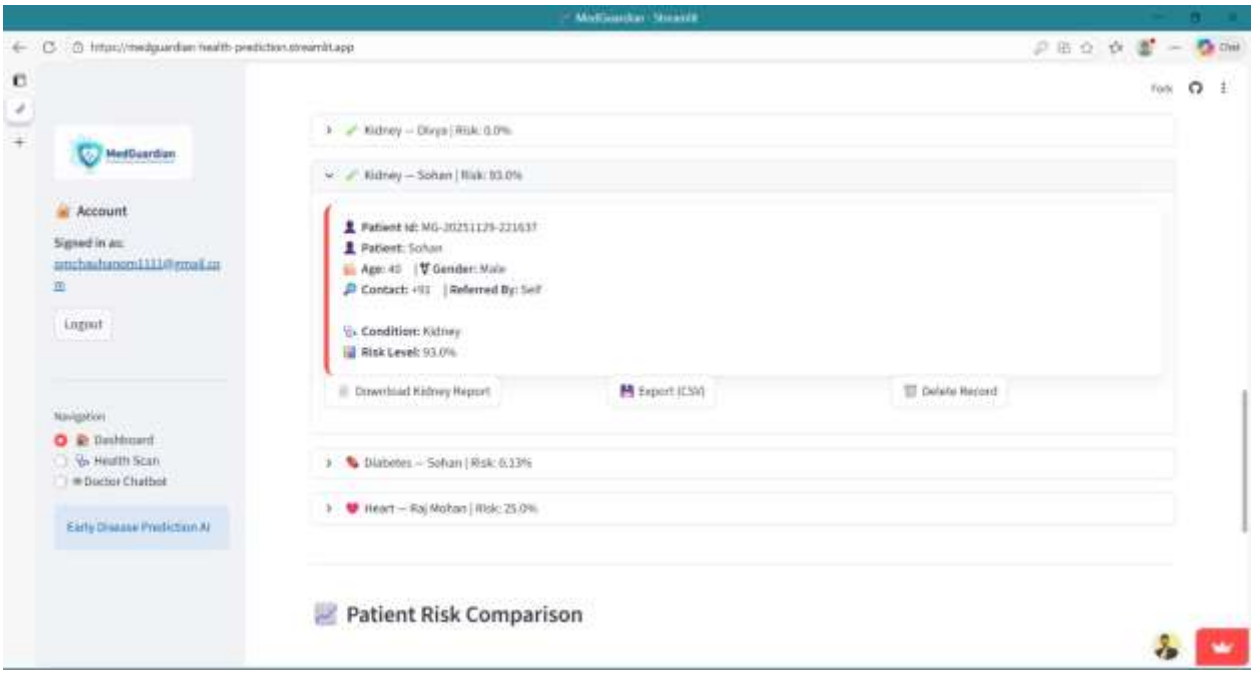

MedGuardian
Account
Logout
Navigation
Dashboard
Health Scan
Doctor Chatbot
Early Disease Prediction AI
Gender
Male
Female
Age
Blood Pressure (mm/Hg)
Specific Gravity
Albumin (0-5)
Sugar (0-5)
Red Blood Cells
Pus Cell
Blood Glucose Random
Blood Urea
Serum Creatinine
Sodium
Potassium
Haemoglobin
Packed Cell Volume


MedGuardian
Account
Logout
Navigation
Dashboard
Health Scan
Doctor Chatbot
Early Disease Prediction AI
Red Blood Cells
Pus Cell
Pus Cell Clumps
Bacteria
Appetite
Pedal Edema
Anaemia
Haemoglobin
Packed Cell Volume
White Blood Cell Count
Red Blood Cell Count
Hypertension
Diabetes Mellitus
Coronary Artery Disease
Predict Kidney Disease

## Chatbot Page:




## Dashboard Details Page:




MedGuardian
Account
Logout
Navigation
Dashboard
Health Scan
Doctor Chatbot
Early Disease Prediction AI
Patient Risk Comparison
Download All Filtered Reports (CSV)

# Report

**MedGuardian**
Contact: +91 9057468262 • Chennai, Tamil Nadu
Email: info@medguardian.com

MedGuardian

## PATIENT MEDICAL REPORT

| | |
|---|---|
| Patient ID | MG-20251201-155627 |
| Patient Name | Dhruv |
| Contact | +91 9057468262 |
| Age | 20 |
| Gender | Male |
| Test Type | Heart |
| Doctor Name | Dr. D. Singh |
| Referred By | Family Doctor |
| Report Date | 01-12-2025 04:01 PM |

### *Test Results*

| Parameter | Value | Normal Range | Status |
|---|---|---|---|
| Resting Blood Pressure | 100 | 90 - 120 | Normal |
| Chest Pain Type | 0 | 0 - 3 | Normal |
| Cholesterol | 220 | < 200 | High |
| Fasting Blood Sugar | 0 | < 100 | Normal |
| Resting ECG | 0 | 0 - 2 | Normal |
| Max Heart Rate | 130 | 60 - 100 | High |
| Exercise Induced Angina | 0 | No | Normal |
| ST Depression | 1 | 0 - 1 | Normal |
| ST Slope | 0 | 0 - 2 | Normal |
| Major Vessels Colored | 1 | 0 - 1 | High |
| Thalassemia | 1 | 0 - 1 | High |



### *AI Diagnosis Summary*

Heart Report

**Om Singh Chauhan**
(AI Generated Signature)

Note: This report is AI generated. Consult a doctor for clinical confirmation.