# AI-Powered Medical Diagnosis System – Documentation

# 1. Project Overview

# The AI-Powered Medical Diagnosis System is a medical web application designed to assist patients and doctors by providing AI-based insights into illnesses using Natural Language Processing (NLP).

# Users can input symptoms, clinical notes, prescriptions, or test results (as text), and the system will generate intelligent predictions about possible illnesses and treatment suggestions.

# The system integrates:

# A frontend for patient and doctor interaction

# A secure backend for data processing and management

# NLP-based machine learning models trained on medical datasets

# Enhanced features like EHR integration, predictive analytics, NLP-powered documentation, and project management tools for clinical workflows

# 2. System Architecture

# Frontend (Client-Side)

# Framework: React / Angular / Vue Features:

# Login/Signup (Patients & Doctors)

# Symptom checker form (text input)

# Upload medical reports (text/PDF converted to text)

# Doctor & Patient dashboards

# Visualizations (charts, confidence scores)

# Centralized dashboard for workflows

# Project & task management

# Backend (Server-Side)

# Framework: Node.js/Express or Django/Flask Responsibilities:

# Authentication & role-based access

# APIs for NLP model communication

# Database operations

# EHR integration (FHIR/HL7 standards)

# Return explainable text-based predictions

# Database

# PostgreSQL/MySQL or MongoDB Stores:

# User profiles

# Patient history

# Uploaded reports & clinical notes

# Prediction logs

# Synced EHR data

# AI/ML Model Layer

# NLP Models (Core Focus):

# BioBERT / ClinicalBERT → process clinical notes, symptoms, prescriptions

# Illness Prediction → multi-class classification on textual medical data

# Predictive Analytics → detect chronic disease trends from historical text data

# Decision Support → suggest potential treatments, medications, and referrals

# Explainability → highlight key terms in text that influenced predictions (e.g., via SHAP/LIME)

# Models are served as REST API / gRPC service.

# 3. Workflow

# User logs in.

# Patient enters symptoms or uploads medical notes (text).

# Backend preprocesses the input and calls the NLP model.

# NLP model returns illness predictions + explanation.

# Backend integrates with EHR and stores results.

# Doctor dashboard displays AI predictions + patient summary.

# Admin uses centralized dashboard for workflow & project management.

# 4. Key Features

# ✅ Symptom-to-Illness prediction (text input only) ✅ Doctor dashboard & patient summary ✅ Explainability (highlight keywords in text) ✅ Secure patient record management ✅ Audit logs for accountability ✅ Centralized dashboard & project tracking ✅ EHR integration ✅ Clinical decision support (NLP-driven) ✅ Predictive analytics on text data ✅ NLP-powered documentation

# 5. Tech Stack

# Frontend: React + Tailwind CSS

# Backend: Flask (Python) / Node.js

# Database: PostgreSQL

# ML Models: PyTorch, Hugging Face Transformers (BioBERT, ClinicalBERT)

# Deployment: Docker + Kubernetes

# Hosting: AWS / GCP / Azure

# 6. Datasets

# Symptom–disease mapping: Kaggle datasets

# Clinical notes: MIMIC-III, PubMed abstracts

# EHR datasets: via FHIR/HL7 standards

# 7. ERD (Entity Relationship Diagram)

# Entities & Relationships:

# Users (Doctors, Patients, Admins)

# Patients → have multiple Records

# Records → include Symptoms, Reports, Clinical Notes

# Predictions → generated by NLP model, linked to Records

# Doctors → review Predictions, add Notes

# Admins → manage tasks via Project Management

# Schema (Simplified):

# Users: (user\_id, name, email, password\_hash, role)

# Patients: (patient\_id, user\_id, dob, gender, medical\_history)

# Records: (record\_id, patient\_id, doctor\_id, symptoms, clinical\_notes, created\_at)

# Predictions: (prediction\_id, record\_id, illness, confidence\_score, explanation, timestamp)

# Tasks: (task\_id, admin\_id, description, status, deadline)

# 8. Future Enhancements

# Real-time chatbot for medical triage (text-only)

# IoT wearable device integration

# Personalized recommendations (medication, lifestyle)

# Multi-language support

# Telemedicine integration

# 10. Demo Design (UI Flow)

# Login Page → Patient/Doctor/Admin roles

# Patient View → Symptom/Notes input → AI prediction results

# Doctor Dashboard → Patient history, predictions, notes

# EHR Integration → Synced health history

# Explainability View → Highlighted text (keywords) + confidence scores

# Admin Dashboard → Task tracking, schedules

# 11. Implementation Plan (Sprints)

**Month 1 – Foundation Setup**

* **Sprint 1 (Weeks 1–2)**
  + Project setup (GitHub repo, CI/CD pipeline)
  + Define system architecture
  + Database schema design (ERD + migrations)
* **Sprint 2 (Weeks 3–4)**
  + Implement authentication (signup/login, roles: patient/doctor/admin)
  + Set up backend API boilerplate (Flask/Node.js)
  + Configure cloud hosting (AWS/GCP/Azure)

**Month 2 – Core Backend & Frontend Skeleton**

* **Sprint 3 (Weeks 5–6)**
  + Build patient dashboard skeleton (symptom input, file upload for reports)
  + Set up doctor dashboard skeleton
  + API for storing medical records
* **Sprint 4 (Weeks 7–8)**
  + Implement patient profile & history management
  + Admin panel skeleton (basic controls)
  + Basic UI/UX design with Tailwind

**Month 3 – AI Integration (Phase 1 – NLP Models)**

* **Sprint 5 (Weeks 9–10)**
  + Integrate NLP model (BioBERT/ClinicalBERT) for text-based symptom analysis
  + Build API wrapper for ML inference
  + Store AI predictions in DB
* **Sprint 6 (Weeks 11–12)**
  + Connect AI prediction results to patient dashboard
  + Doctor dashboard displays patient AI results
  + Implement explainability (confidence scores, SHAP basics)

**Month 4 – AI Integration (Phase 2 – Enhanced NLP)**

* **Sprint 7 (Weeks 13–14)**
  + Expand NLP to include **clinical notes + lab reports parsing**
  + Add medical entity extraction (diagnosis, symptoms, medications)
* **Sprint 8 (Weeks 15–16)**
  + Multi-input predictions (symptoms + lab reports + doctor notes)
  + Enhance doctor dashboard with “AI summary” view
  + Secure role-based access enforcement

**Month 5 – EHR Integration**

* **Sprint 9 (Weeks 17–18)**
  + Research and connect FHIR/HL7 API for external EHR
  + Sync patient history into system
* **Sprint 10 (Weeks 19–20)**
  + Enable EHR import/export
  + Display full medical history in patient profile
  + Add audit logging for compliance

**Month 6 – Dashboard & Workflow Management**

* **Sprint 11 (Weeks 21–22)**
  + Centralized dashboard for Admin
  + Role-based quick navigation (doctor/patient/admin views)
* **Sprint 12 (Weeks 23–24)**
  + Implement project & task management features (assignments, deadlines, statuses)
  + Calendar view for tasks

**Month 7 – Predictive Analytics & Clinical Support**

* **Sprint 13 (Weeks 25–26)**
  + Predictive analytics using NLP models for chronic illness risk (diabetes, heart disease)
  + Patient risk monitoring
* **Sprint 14 (Weeks 27–28)**
  + Clinical decision support (treatment suggestions, evidence references)
  + Improve explainability with SHAP/LIME visualizations

**Month 8 – Testing & Security**

* **Sprint 15 (Weeks 29–30)**
  + Security hardening (data encryption, JWT auth, GDPR/HIPAA compliance)
  + Load testing & API optimization
* **Sprint 16 (Weeks 31–32)**
  + End-to-end system testing
  + Collect feedback from test users (doctors/patients)
  + Bug fixes

**Month 9 – Finalization & Deployment**

* **Sprint 17 (Weeks 33–34)**
  + Deploy stable version to production (cloud hosting, Docker/Kubernetes)
  + Build documentation (API docs, user guide)
* **Sprint 18 (Weeks 35–36)**
  + Final polish of dashboards (UI/UX improvements)
  + Demo preparation (presentations, test case walkthroughs)
  + Future roadmap planning (telemedicine, chatbot integration)