Health Prediction System: Full Project Draft

1. Project Goal

- Predict a user's **disease category** based on vital signs and symptoms.
- Input features:
    - **From message:** temperature, heart_rate, pulse, bp_sys, bp_dia, humidity
    - **From HTML form/user input:** fever, cough, chest_pain, shortness_of_breath, fatigue, headache
- Output: disease prediction (e.g., Normal, Heart_Risk, Hypertension, Hypotension, Fever_Respiratory)
- Optional: Suggest drugs or precautions based on predicted disease.

2. Data Collection

- Collect dataset with **all 12 input features + disease label**.
- Ensure:
    - No duplicate rows.
    - No missing values.
    - Sufficient examples for each disease category.
- Example dataset columns:

| temperature | heart_rate | pulse | bp_sys | bp_dia | humidity | fever | cough | chest_pain | shortness_of_breath | fatigue | headache | disease |

3. Data Preprocessing

- **Normalize/scale numeric features** (temperature, heart_rate, pulse, bp_sys, bp_dia, humidity) to match model training.
- **Encode categorical features**:
    - fever, cough, chest_pain, shortness_of_breath, fatigue, headache → 0 or 1
    - disease → one-hot encoding or label encoding
- **Split dataset**:
    - Training: 70–80%

  - Validation: 10–15%
  - Test: 10–15%

4. Model Selection

- **Machine Learning Options:**
  - Random Forest
  - XGBoost
  - Support Vector Machine (SVM)
- **Deep Learning Option:**
  - Multi-layer Perceptron (MLP) with:
    - Input layer: 12 nodes (features)
    - Hidden layers: 2–3 layers, 32–128 neurons each, ReLU activation
    - Output layer: number of disease categories, softmax activation
- Evaluate performance on validation set using:
  - Accuracy
  - F1-score
  - Confusion matrix

5. Backend: FastAPI

- **Receive inputs** (JSON) from:
  1. **Message**: temperature, heart_rate, pulse, bp_sys, bp_dia, humidity
  2. **HTML form**: fever, cough, chest_pain, shortness_of_breath, fatigue, headache
- **Combine features** into 12-element vector.
- Feed vector into trained ML/DL model.
- Return predicted disease + drug suggestions.

FastAPI Example Structure:

```
from fastapi import FastAPI

from pydantic import BaseModel

import joblib, numpy as np
```

```
app = FastAPI()

class UserInput(BaseModel):
    temperature: float
    heart_rate: float
    pulse: float
    bp_sys: float
    bp_dia: float
    humidity: float
    fever: int
    cough: int
    chest_pain: int
    shortness_of_breath: int
    fatigue: int
    headache: int

model = joblib.load("disease_model.pkl")

drug_mapping = {
    "Heart_Risk": "Consult cardiologist; Beta blockers; ACE inhibitors",
    "Fever_Respiratory": "Paracetamol; Cough syrup; Consult physician",
    "Hypertension": "Amlodipine; Lifestyle changes",
    "Hypotension": "Increase fluids; Monitor BP",
    "Normal": "No action"
}

@app.post("/predict")
def predict(input_data: UserInput):
    features = [
        input_data.temperature,
        input_data.heart_rate,
        input_data.pulse,
        input_data.bp_sys,
        input_data.bp_dia,
        input_data.humidity,
        input_data.fever,
        input_data.cough,
        input_data.chest_pain,
        input_data.shortness_of_breath,
        input_data.fatigue,
        input_data.headache
    ]
    features_array = np.array(features).reshape(1, -1)
    disease_pred = model.predict(features_array)[0]
    suggested_drugs = drug_mapping.get(disease_pred, "Consult doctor")
    return {"disease": disease_pred, "suggested_drugs": suggested_drugs}
```

6. Frontend: HTML / JS Form

- Collect direct symptom input from user.
- Send JSON to FastAPI endpoint.
- Receive and display predicted disease + drug suggestions.

```
<form id="symptomForm">
  <!-- Checkbox inputs for each symptom -->
  <button type="submit">Submit</button>
</form>


<script>
```

```
document.getElementById('symptomForm').addEventListener('submit', async (e) => {
  e.preventDefault();
  const data = { /* collect form + message features */ };
  const response = await fetch('http://localhost:8000/predict', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify(data)
  });
  const result = await response.json();
  alert(`Disease: ${result.disease}\nDrugs: ${result.suggested_drugs}`);
});
</script>
```

7. Message Collection (Optional)

- Use **Telegram Bot** (recommended) or WhatsApp API.
- Bot collects:
    - temperature, heart_rate, pulse, bp_sys, bp_dia, humidity
- Sends JSON to FastAPI backend.

8. Workflow Summary

1. **User sends message** → bot extracts vital signs.
2. **User fills web form** → collects symptoms.
3. **Combine features** → 12-element vector.
4. **Send JSON to FastAPI** → backend calls ML/DL model.
5. **Model predicts disease** → return prediction + drug suggestions.
6. **Display result** to user.

9. Optional Enhancements

- Input validation for ranges (temperature 35–42, heart rate 40–180, etc.)
- Logging and storing user data for model improvement.
- Deploy as web app with Docker + cloud server.
- Add explanations for predictions using SHAP/LIME for ML models.

User Message
(Telegram/WA)
temperature,
heart_rate,
pulse, bp_sys,
bp_dia, humidity

HTML / JS Form
(Symptoms Input)
fever, cough,
chest_pain,
shortness_of_breath,
fatigue, headache

Combine Features
(12 total)

FastAPI Backend
- Receive JSON
- Preprocess
- Feed to ML/DL

ML/DL Model
- Predict disease
- Multi-class

Drug Suggestion
Mapping Table
Based on Disease

Return Result
- Predicted Disease
- Suggested Drugs

User Output
- Alert / Display
- Web or Message