```
from flask import Flask, request, render_template_string, redirect, session, url_for, jsonify
import pandas as pd
import os
from datetime import datetime, timedelta, date
import numpy as np
from threading import Thread
from flask_dance.contrib.google import make_google_blueprint, google
from PIL import Image
import pytesseract

from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.ensemble import VotingClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import classification_report, accuracy_score
import warnings
warnings.filterwarnings('ignore')

app = Flask(__name__)
app.secret_key = "your_secret_key"  # required for Flask sessions
CSV_FILE = "thalassemia_patients.csv"
HISTORY_FILE = "history.csv"
URGENCY_FILE = "urgency.csv"
BLOOD_GROUPS = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']

# Create static folder if not exists
```

```
os.makedirs("static", exist_ok=True)


# Initialize CSV files with correct structure
def initialize_csv_files():
    columns = [
        "Name", "DOB", "Phone", "Aadhaar", "State", "City",
        "Last_Transfusion", "Next_Transfusion", "Hb", "Iron", "Weight", "Blood_Group", "Timestamp",
"Email"
    ]


    if not os.path.exists(CSV_FILE):
        pd.DataFrame(columns=columns).to_csv(CSV_FILE, index=False)


    if not os.path.exists(HISTORY_FILE):
        pd.DataFrame(columns=columns + ["Urgency"]).to_csv(HISTORY_FILE, index=False)


    if not os.path.exists(URGENCY_FILE):
        pd.DataFrame(columns=columns + ["Urgency", "Predicted_Score"]).to_csv(URGENCY_FILE,
index=False)


initialize_csv_files()


def clean_csv_duplicates():
    """Remove duplicate columns from existing CSV if they exist"""
    for file in [CSV_FILE, HISTORY_FILE, URGENCY_FILE]:
        try:
            if os.path.exists(file):
                df = pd.read_csv(file)
                df = df.loc[:, ~df.columns.duplicated()]
                df.to_csv(file, index=False)
                print(f"✅ {file} cleaned of duplicate columns")
        except Exception as e:
```

```
        print(f" ⚠️ Error cleaning {file}: {e}")

clean_csv_duplicates()

def calculate_urgency_level(row):
    """Calculate urgency level based on clinical parameters"""
    score = 0

    # Hemoglobin urgency (most critical factor)
    hb = float(row.get('Hb', 0)) if pd.notna(row.get('Hb')) else 0
    if hb < 6: score += 5  # Life-threatening
    elif hb < 7: score += 4  # Critical
    elif hb < 8: score += 3  # High
    elif hb < 9: score += 2  # Moderate
    elif hb < 10: score += 1  # Low

    # Days since last transfusion
    try:
        if pd.notna(row.get('Last_Transfusion')):
            last_transfusion = pd.to_datetime(row['Last_Transfusion'], errors='coerce')
            if pd.notna(last_transfusion):
                days_since_last = (datetime.now() - last_transfusion).days
                if days_since_last > 45: score += 3  # Very overdue
                elif days_since_last > 35: score += 2  # Overdue
                elif days_since_last > 28: score += 1  # Due soon
    except:
        pass

    # Days to next transfusion (overdue patients)
    try:
        if pd.notna(row.get('Next_Transfusion')):
```

```
        next_transfusion = pd.to_datetime(row['Next_Transfusion'], errors='coerce')
        if pd.notna(next_transfusion):
            days_to_next = (next_transfusion - datetime.now()).days
            if days_to_next < -7: score += 4  # Severely overdue
            elif days_to_next < -3: score += 3  # Overdue
            elif days_to_next < 0: score += 2  # Past due
            elif days_to_next <= 2: score += 1  # Due very soon
    except:
        pass

    # Iron overload risk
    iron = float(row.get('Iron', 0)) if pd.notna(row.get('Iron')) else 0
    if iron > 2000: score += 2  # High iron toxicity risk
    elif iron > 1000: score += 1  # Moderate iron overload

    # Weight considerations (very low weight indicates severity)
    weight = float(row.get('Weight', 0)) if pd.notna(row.get('Weight')) else 0
    if weight < 30: score += 1  # Underweight concern

    # Convert score to urgency level
    if score >= 8: return 3, "Critical"
    elif score >= 5: return 2, "High"
    elif score >= 2: return 1, "Moderate"
    else: return 0, "Low"

def remove_duplicates_from_dataframe(df):
    """Remove duplicate entries based on Aadhaar and Phone, keeping the latest"""
    print(f"🔍 Before deduplication: {len(df)} records")

    # Sort by timestamp to ensure latest records are kept
    df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')
```

```
    df = df.sort_values('Timestamp', ascending=False)

    # Remove duplicates based on Aadhaar (primary) and Phone (secondary)
    df_dedup = df.drop_duplicates(subset=['Aadhaar'], keep='first')

    # If still duplicates by phone, remove those too
    df_dedup = df_dedup.drop_duplicates(subset=['Phone'], keep='first')

    duplicates_removed = len(df) - len(df_dedup)
    if duplicates_removed > 0:
        print(f"🗑️ Removed {duplicates_removed} duplicate records")

    print(f"✅ After deduplication: {len(df_dedup)} unique records")
    return df_dedup

def train_and_predict_urgency(patient_aadhaar=None, patient_phone=None):
    """Train ML models and predict urgency for patients with detailed debugging"""
    try:
        print("\n" + "="*80)
        print("🧠 STARTING ML TRAINING AND PREDICTION")
        print("="*80)

        # Load history data for training
        history_df = pd.read_csv(HISTORY_FILE)
        current_df = pd.read_csv(CSV_FILE)

        print(f"📊 DATASET OVERVIEW:")
        print(f"   - History records: {len(history_df)}")
        print(f"   - Current patients: {len(current_df)}")

        if history_df.empty:
```

```
    print(" ⚠️ No historical data available for training. Using rule-based approach.")
    return predict_urgency_rule_based()

# Prepare training data from history
history_df = history_df.dropna(subset=["Hb", "Iron", "Weight", "Blood_Group"])

print(f"📈 TRAINING DATA AFTER CLEANING:")
print(f"   - Valid training records: {len(history_df)}")

if len(history_df) < 5:  # Need minimum data for training
    print(" ⚠️ Insufficient historical data for ML training. Using rule-based approach.")
    return predict_urgency_rule_based()

# Show training data details
print(f"\n📋 TRAINING DATA BREAKDOWN:")
urgency_counts = history_df['Urgency'].value_counts()
for urgency, count in urgency_counts.items():
    print(f"   - {urgency}: {count} patients")

print(f"\n🔬 CLINICAL PARAMETERS IN TRAINING DATA:")
print(f"   - Hb levels: min={history_df['Hb'].min():.1f}, max={history_df['Hb'].max():.1f}, avg={history_df['Hb'].mean():.1f}")
print(f"   - Iron levels: min={history_df['Iron'].min():.1f}, max={history_df['Iron'].max():.1f}, avg={history_df['Iron'].mean():.1f}")
print(f"   - Weight: min={history_df['Weight'].min():.1f}, max={history_df['Weight'].max():.1f}, avg={history_df['Weight'].mean():.1f}")

# Feature engineering for history data
history_features = prepare_features(history_df)

print(f"\n🛠️ FEATURE ENGINEERING:")
print(f"   - Features used: {list(history_features.columns)}")
```

```
print(f"   - Feature matrix shape: {history_features.shape}")

# Get labels from history (convert text to numeric)
urgency_map = {"Low": 0, "Moderate": 1, "High": 2, "Critical": 3}
history_df['Urgency_Numeric'] = history_df['Urgency'].map(urgency_map).fillna(0)

X_train = history_features
y_train = history_df['Urgency_Numeric'].values

print(f"\n🎯 TRAINING TARGET DISTRIBUTION:")
unique, counts = np.unique(y_train, return_counts=True)
for u, c in zip(unique, counts):
    urgency_text = {0: "Low", 1: "Moderate", 2: "High", 3: "Critical"}[u]
    print(f"   - {urgency_text} ({u}): {c} samples")

# Train ensemble model
print(f"\n⏳ TRAINING ENSEMBLE MODEL...")
model, train_accuracy = train_ensemble_model(X_train, y_train)
print(f"✅ Model training completed with accuracy: {train_accuracy:.3f}")

# Predict for current patients
current_df = current_df.dropna(subset=["Hb", "Iron", "Weight", "Blood_Group"])

print(f"\n🔮 MAKING PREDICTIONS:")
print(f"   - Patients to predict: {len(current_df)}")

if not current_df.empty:
    # Remove duplicates before prediction
    current_df = remove_duplicates_from_dataframe(current_df)

    current_features = prepare_features(current_df)
```

```
        predictions = model.predict(current_features)

        print(f"\n📊 PREDICTION RESULTS:")
        pred_unique, pred_counts = np.unique(predictions, return_counts=True)
        for p, c in zip(pred_unique, pred_counts):
            urgency_text = {0: "Low", 1: "Moderate", 2: "High", 3: "Critical"}[p]
            print(f"   - {urgency_text}: {c} patients")

        # Convert predictions back to text
        reverse_urgency_map = {0: "Low", 1: "Moderate", 2: "High", 3: "Critical"}
        current_df['Urgency'] = [reverse_urgency_map[pred] for pred in predictions]
        current_df['Predicted_Score'] = predictions

        # Show individual predictions for critical/high urgency patients
        critical_high = current_df[current_df['Urgency'].isin(['Critical', 'High'])]
        if not critical_high.empty:
            print(f"\n🚨 HIGH PRIORITY PATIENTS:")
            for _, patient in critical_high.iterrows():
                print(f"   - {patient['Name']} (Aadhaar: {patient['Aadhaar']}): {patient['Urgency']} (Hb: {patient['Hb']}, Iron: {patient['Iron']}, Weight: {patient['Weight']})")

        # Update urgency file with deduplication
        update_urgency_file(current_df)

    print(f"\n✅ ML-based urgency prediction completed successfully!")
    print("="*80 + "\n")

  except Exception as e:
    print(f"⚠️ Error in ML prediction: {e}")
    predict_urgency_rule_based()
```

```
def prepare_features(df):
    """Prepare features for ML model"""
    features = df.copy()

    # Blood group rarity weights
    rarity_weights = {
        "AB-": 0.1, "B-": 0.2, "A-": 0.3, "O-": 0.4,
        "AB+": 0.5, "B+": 0.6, "A+": 0.7, "O+": 0.8
    }
    features["Blood_Weight"] = features["Blood_Group"].map(rarity_weights).fillna(0.5)

    # Calculate days since last transfusion
    features["days_since_last"] = features.apply(lambda row:
        (datetime.now() - pd.to_datetime(row['Last_Transfusion'], errors='coerce')).days
        if pd.notna(row.get('Last_Transfusion')) else 30, axis=1)

    # Calculate days to next transfusion
    features["days_to_next"] = features.apply(lambda row:
        (pd.to_datetime(row['Next_Transfusion'], errors='coerce') - datetime.now()).days
        if pd.notna(row.get('Next_Transfusion')) else 30, axis=1)

    # Select numerical features
    feature_columns = ["Hb", "Iron", "Weight", "Blood_Weight", "days_since_last", "days_to_next"]
    X = features[feature_columns].fillna(0)

    return X

def train_ensemble_model(X_train, y_train):
    """Train ensemble model with multiple algorithms and return accuracy"""
    try:
        # Split for validation
```

```
    X_tr, X_val, y_tr, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42,
stratify=y_train)

    print(f"   - Training set: {X_tr.shape[0]} samples")
    print(f"   - Validation set: {X_val.shape[0]} samples")

    # Scale features
    scaler = RobustScaler()
    X_tr_scaled = scaler.fit_transform(X_tr)
    X_val_scaled = scaler.transform(X_val)

    # Define individual models
    models = [
        ("xgb", XGBClassifier(verbosity=0, use_label_encoder=False, random_state=42)),
        ("cat", CatBoostClassifier(verbose=0, random_state=42)),
        ("lr", LogisticRegression(max_iter=1000, random_state=42)),
        ("gb", GradientBoostingClassifier(random_state=42)),
        ("svm", SVC(probability=True, random_state=42)),
        ("nb", GaussianNB()),
        ("mlp", MLPClassifier(max_iter=500, random_state=42))
    ]

    print(f"   - Using {len(models)} different algorithms in ensemble")

    # Create ensemble
    ensemble = VotingClassifier(estimators=models, voting="soft")
    ensemble.fit(X_tr_scaled, y_tr)

    # Validate model
    val_predictions = ensemble.predict(X_val_scaled)
    accuracy = accuracy_score(y_val, val_predictions)
```

```
    print(f"   - Validation accuracy: {accuracy:.3f}")

    # Show classification report
    print(f"\n📈 DETAILED PERFORMANCE REPORT:")
    report = classification_report(y_val, val_predictions,
                                   target_names=["Low", "Moderate", "High", "Critical"],
                                   zero_division=0)
    for line in report.split('\n'):
        if line.strip():
            print(f"   {line}")

    # Create a wrapper that includes scaling
    class ScaledEnsemble:
        def __init__(self, scaler, model):
            self.scaler = scaler
            self.model = model

        def predict(self, X):
            X_scaled = self.scaler.transform(X)
            return self.model.predict(X_scaled)

    return ScaledEnsemble(scaler, ensemble), accuracy

except Exception as e:
    print(f"Error in ensemble training: {e}")
    # Fallback to simple model
    from sklearn.ensemble import RandomForestClassifier
    model = RandomForestClassifier(random_state=42)
    model.fit(X_train, y_train)
    return model, 0.0
```

```
def predict_urgency_rule_based():
    """Fallback rule-based urgency prediction"""
    try:
        print("\n🔧 USING RULE-BASED URGENCY PREDICTION")
        df = pd.read_csv(CSV_FILE)
        df = df.dropna(subset=["Hb", "Iron", "Weight", "Blood_Group"])

        # Remove duplicates before rule-based prediction
        df = remove_duplicates_from_dataframe(df)

        print(f"📊 Applying rules to {len(df)} patients")

        urgency_results = []
        for _, row in df.iterrows():
            score, urgency_text = calculate_urgency_level(row)
            urgency_results.append({
                **row.to_dict(),
                'Urgency': urgency_text,
                'Predicted_Score': score
            })
            print(f"   - {row['Name']}: {urgency_text} (Score: {score})")

        if urgency_results:
            urgency_df = pd.DataFrame(urgency_results)
            update_urgency_file(urgency_df)

        print("✅ Rule-based urgency prediction completed")

    except Exception as e:
        print(f"⚠️ Error in rule-based prediction: {e}")
```

```
def update_urgency_file(df):
    """Update urgency file and remove expired patients with deduplication"""
    try:
        print(f"\n 📝 UPDATING URGENCY FILE:")

        # Remove duplicates first
        df = remove_duplicates_from_dataframe(df)

        # Filter patients within urgency window (next transfusion ± 5 days)
        df["Next_Transfusion"] = pd.to_datetime(df["Next_Transfusion"], errors='coerce')
        today = pd.to_datetime(datetime.today().date())

        # Include patients whose next transfusion is within 5 days past or future
        urgency_window = df[
            df["Next_Transfusion"].between(
                today - timedelta(days=5),
                today + timedelta(days=5)
            )
        ]

        print(f"   - Patients in 5-day urgency window: {len(urgency_window)}")

        if not urgency_window.empty:
            # Sort by urgency (Critical first) then by Hb level (lowest first)
            urgency_order = {"Critical": 0, "High": 1, "Moderate": 2, "Low": 3}
            urgency_window["urgency_rank"] = urgency_window["Urgency"].map(urgency_order)
            urgency_window = urgency_window.sort_values(["urgency_rank", "Hb"])
            urgency_window = urgency_window.drop("urgency_rank", axis=1)

            # Final deduplication check before saving
```

```
        urgency_window = remove_duplicates_from_dataframe(urgency_window)

        urgency_window.to_csv(URGENCY_FILE, index=False)
        print(f"✅ Urgency file updated with {len(urgency_window)} unique patients")

        # Show urgency breakdown
        urgency_counts = urgency_window['Urgency'].value_counts()
        for urgency, count in urgency_counts.items():
            print(f"   - {urgency}: {count} patients")

    else:
        # Create empty urgency file
        pd.DataFrame(columns=pd.read_csv(CSV_FILE).columns.tolist() + ["Urgency",
"Predicted_Score"]).to_csv(URGENCY_FILE, index=False)
        print("✅ No patients in urgency window - empty urgency file created")

  except Exception as e:
    print(f"⚠️ Error updating urgency file: {e}")

def add_to_history(patient_data, urgency="Low"):
  """Add patient data to history file"""
  try:
    history_df = pd.read_csv(HISTORY_FILE)
    patient_data_with_urgency = patient_data.copy()
    patient_data_with_urgency['Urgency'] = urgency

    print(f"📚 Adding patient {patient_data.get('Name', 'Unknown')} to history with urgency:
{urgency}")

    history_df = pd.concat([history_df, pd.DataFrame([patient_data_with_urgency])],
ignore_index=True)
    history_df.to_csv(HISTORY_FILE, index=False)
```

```
        print(" ✅ Patient data added to history")
    except Exception as e:
        print(f" ⚠️ Error adding to history: {e}")

def check_patient_exists(aadhaar=None, phone=None):
    """Check if patient exists in thalassemia_patients.csv"""
    try:
        df = pd.read_csv(CSV_FILE)
        if aadhaar:
            aadhaar_match = df[df["Aadhaar"].astype(str) == str(aadhaar)]
            if not aadhaar_match.empty:
                return True, aadhaar_match.iloc[-1]

        if phone:
            phone_match = df[df["Phone"].astype(str) == str(phone)]
            if not phone_match.empty:
                return True, phone_match.iloc[-1]

        return False, None
    except Exception as e:
        print(f"Error checking patient existence: {e}")
        return False, None

# ------------------------------------ ADMIN ------------------------------------

@app.route("/admin")
def admin():
    return render_template_string('''
        <h1>Admin Panel</h1>
        <a href="/admin/create">➕ Create New Patient</a><br><br>
        <a href="/admin/list">📋 Registered Patient List</a><br><br>
```

```
        <a href="/admin/urgency">🚨 View Urgency Table</a><br><br>
        <a href="/admin/history">📊 View Patient History</a><br><br>
        <a href="/admin/stats">📈 System Statistics</a>
    ''')

@app.route("/admin/create", methods=["GET", "POST"])
def create_patient():
    today = date.today().isoformat()
    if request.method == "POST":
        aadhaar = request.form["aadhaar"]
        phone = request.form["phone"]

        # Check if patient already exists
        exists, existing_patient = check_patient_exists(aadhaar=aadhaar, phone=phone)
        if exists:
            return f"❌ Error: Patient with Aadhaar {aadhaar} or Phone {phone} already exists! <a
href='/admin/create'>Back</a> | <a href='/admin/list'>View Patient List</a>"

        next_transfusion_str = request.form["next_transfusion"]
        if next_transfusion_str:
            next_transfusion_date = datetime.strptime(next_transfusion_str, "%Y-%m-%d").date()
            if next_transfusion_date < date.today():
                return "❌ Error: Next transfusion date cannot be in the past. <a
href='/admin/create'>Back</a>"

        data = {
            "Name": request.form["name"],
            "DOB": request.form["dob"],
            "Phone": phone,
            "Aadhaar": aadhaar,
            "State": request.form["state"],
```

```
    "City": request.form["city"],
    "Last_Transfusion": request.form["last_transfusion"],
    "Next_Transfusion": request.form["next_transfusion"],
    "Hb": request.form["hb"],
    "Iron": request.form["iron"],
    "Weight": request.form["weight"],
    "Blood_Group": request.form["blood_group"],
    "Timestamp": datetime.now(),
    "Email": ""
}

# Add to main CSV
df = pd.read_csv(CSV_FILE)
df = pd.concat([df, pd.DataFrame([data])], ignore_index=True)
df.to_csv(CSV_FILE, index=False)

# Check if patient has complete medical vitals for training
has_complete_vitals = all([
    data["Hb"], data["Iron"], data["Weight"],
    data["Blood_Group"], data["Next_Transfusion"], data["Last_Transfusion"]
])

if has_complete_vitals:
    # Calculate urgency
    score, urgency_text = calculate_urgency_level(data)
    # Add to history
    add_to_history(data, urgency_text)
    # Trigger ML training and prediction
    train_and_predict_urgency(patient_aadhaar=aadhaar)
else:
    print(" ⚠️ Patient registered but missing complete medical vitals for ML training")
```

```
    return " ✅ Patient record added successfully! <a href='/admin'>Back to Admin</a> | <a
href='/admin/create'>Add Another Patient</a>"


  return render_template_string('''
    <h2>Create New Patient</h2>
    <form method="POST">
      Name: <input name="name" required><br><br>
      DOB: <input name="dob" type="date" required><br><br>
      Phone: <input name="phone" pattern="\\d{10}" required><br><br>
      Aadhaar: <input name="aadhaar" pattern="\\d{12}" required><br><br>
      State: <input name="state" required><br><br>
      City: <input name="city" required><br><br>
      Last Transfusion: <input name="last_transfusion" type="date"><br><br>
      Next Transfusion: <input name="next_transfusion" type="date" min="{{ today }}"><br><br>
      Hb: <input name="hb" step="0.1" type="number"><br><br>
      Iron: <input name="iron" step="0.1" type="number"><br><br>
      Weight: <input name="weight" step="0.1" type="number"><br><br>
      Blood Group:
      <select name="blood_group" required>
        <option value="">Select Blood Group</option>
        {% for bg in blood_groups %}
          <option value="{{bg}}">{{bg}}</option>
        {% endfor %}
      </select><br><br>
      <input type="submit" value=" ✅ Register Patient">
    </form>
  ''', blood_groups=BLOOD_GROUPS, today=today)


@app.route('/admin/list', methods=['GET', 'POST'])
def list_patients():
```

```
df = pd.read_csv(CSV_FILE)

if request.method == 'POST':
    query = request.form.get('search', '')
    search_result = df[
        df["Name"].str.contains(query, case=False, na=False) |
        df["Phone"].astype(str).str.contains(query, na=False) |
        df["Aadhaar"].astype(str).str.contains(query, na=False)
    ]
else:
    search_result = df

html_template = '''
<!doctype html>
<html>
<head>
    <title>Patient List</title>
    <style>
        table { border-collapse: collapse; width: 100%; font-family: Arial, sans-serif; }
        th, td { border: 1px solid #ddd; padding: 8px; }
        th { background-color: #f2f2f2; }
        input[type=text] { width: 300px; padding: 6px; margin: 10px 0; }
        button { padding: 6px 12px; }
    </style>
</head>
<body>
    <h2>Thalassemia Patients List ({{ search_result|length }} records)</h2>
    <form method="POST" action="/admin/list">
        <input type="text" name="search" placeholder="Search by Name / Phone / Aadhaar">
        <button type="submit">Search</button>
    </form>
```

```
        <table>
            <thead>
                <tr>
                    {% for col in patients[0].keys() if patients %}
                    <th>{{ col }}</th>
                    {% endfor %}
                </tr>
            </thead>
            <tbody>
                {% for patient in patients %}
                <tr>
                    {% for value in patient.values() %}
                    <td>{{ value }}</td>
                    {% endfor %}
                </tr>
                {% endfor %}
            </tbody>
        </table>
        <br><a href="/admin">🔙 Back to Admin</a>
    </body>
    </html>
    '''
    return render_template_string(html_template, patients=search_result.to_dict(orient='records'),
search_result=search_result)


@app.route("/admin/urgency")
def view_urgency():
    try:
        df = pd.read_csv(URGENCY_FILE)
        if df.empty:
            return "⚠️ No urgent patients found. <a href='/admin'>🔙 Back to Admin</a>"
```

```
html_template = '''
<h2> 🩸 Urgency Table - Next 5 Days ({{ df|length }} unique patients)</h2>
<style>
  table { border-collapse: collapse; width: 100%; font-family: Arial, sans-serif; }
  th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
  th { background-color: #f2f2f2; font-weight: bold; }
  .critical { background-color: #ffebee; color: #d32f2f; font-weight: bold; }
  .high { background-color: #fff3e0; color: #f57c00; font-weight: bold; }
  .moderate { background-color: #fffde7; color: #f9a825; }
  .low { background-color: #e8f5e8; color: #388e3c; }
</style>
<table>
  <thead>
    <tr>
      {% for col in df.columns %}
      <th>{{ col.replace('_', ' ').title() }}</th>
      {% endfor %}
    </tr>
  </thead>
  <tbody>
    {% for _, row in df.iterrows() %}
    <tr class="{{ row.get('Urgency', '').lower() }}">
      {% for value in row %}
      <td>{{ value }}</td>
      {% endfor %}
    </tr>
    {% endfor %}
  </tbody>
</table>
<br><a href="/admin">🔙 Back to Admin</a>
```

```
        '''

        return render_template_string(html_template, df=df)

    except FileNotFoundError:
        return " ⚠️ No urgency data found. <a href='/admin'> 🔙 Back to Admin</a>"
    except Exception as e:
        return f" ⚠️ Error loading urgency data: {str(e)} <a href='/admin'> 🔙 Back to Admin</a>"

@app.route("/admin/history", methods=["GET", "POST"])
def view_history():
    try:
        df = pd.read_csv(HISTORY_FILE)
        if df.empty:
            return " ⚠️ No historical data found. <a href='/admin'> 🔙 Back to Admin</a>"

        # Handle search functionality
        search_query = ""
        if request.method == 'POST':
            search_query = request.form.get('search', '').strip()
            if search_query:
                df = df[
                    df["Name"].str.contains(search_query, case=False, na=False) |
                    df["Phone"].astype(str).str.contains(search_query, na=False) |
                    df["Aadhaar"].astype(str).str.contains(search_query, na=False) |
                    df["Urgency"].str.contains(search_query, case=False, na=False)
                ]

        # Calculate statistics
        total_records = len(df)
        urgency_stats = df['Urgency'].value_counts().to_dict() if 'Urgency' in df.columns else {}
```

```
# Get average clinical parameters
avg_hb = df['Hb'].mean() if 'Hb' in df.columns and not df['Hb'].isna().all() else 0
avg_iron = df['Iron'].mean() if 'Iron' in df.columns and not df['Iron'].isna().all() else 0
avg_weight = df['Weight'].mean() if 'Weight' in df.columns and not df['Weight'].isna().all() else 0

html_template = '''
<!doctype html>
<html>
<head>
  <title>Patient History</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    table { border-collapse: collapse; width: 100%; margin-top: 20px; }
    th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
    th { background-color: #f2f2f2; font-weight: bold; }
    .stats-container { display: flex; gap: 20px; margin: 20px 0; }
    .stat-box { background: #f8f9fa; padding: 15px; border-radius: 5px; border-left: 4px solid #007bff; }
    .critical { background-color: #ffebee; color: #d32f2f; font-weight: bold; }
    .high { background-color: #fff3e0; color: #f57c00; font-weight: bold; }
    .moderate { background-color: #fffde7; color: #f9a825; }
    .low { background-color: #e8f5e8; color: #388e3c; }
    input[type=text] { width: 300px; padding: 8px; margin: 10px 0; }
    button { padding: 8px 15px; background: #007bff; color: white; border: none; border-radius: 3px; }
  </style>
</head>
<body>
  <h2> 📊 Patient History & Training Data</h2>

  <div class="stats-container">
```

```
        <div class="stat-box">
            <h4>📈 Total Records</h4>
            <p><strong>{{ total_records }}</strong> patient records</p>
        </div>
        <div class="stat-box">
            <h4>🩸 Average Clinical Values</h4>
            <p>Hb: <strong>{{ "%.1f"|format(avg_hb) }}</strong> g/dL</p>
            <p>Iron: <strong>{{ "%.1f"|format(avg_iron) }}</strong> µg/dL</p>
            <p>Weight: <strong>{{ "%.1f"|format(avg_weight) }}</strong> kg</p>
        </div>
        <div class="stat-box">
            <h4>🚨 Urgency Distribution</h4>
            {% for urgency, count in urgency_stats.items() %}
            <p>{{ urgency }}: <strong>{{ count }}</strong></p>
            {% endfor %}
        </div>
    </div>

    <form method="POST" action="/admin/history">
        <input type="text" name="search" value="{{ search_query }}" placeholder="Search by Name, Phone, Aadhaar, or Urgency">
        <button type="submit">🔍 Search</button>
        {% if search_query %}
        <a href="/admin/history" style="margin-left: 10px; text-decoration: none; color: #6c757d;">Clear Search</a>
        {% endif %}
    </form>

    {% if df|length > 0 %}
    <p><strong>Showing {{ df|length }} records</strong> {% if search_query %}(filtered by "{{ search_query }}"){% endif %}</p>
```

```
    <table>
      <thead>
        <tr>
          {% for col in df.columns %}
          <th>{{ col.replace('_', ' ').title() }}</th>
          {% endfor %}
        </tr>
      </thead>
      <tbody>
        {% for _, row in df.iterrows() %}
        <tr class="{{ row.get('Urgency', '').lower() }}">
          {% for value in row %}
          <td>{{ value }}</td>
          {% endfor %}
        </tr>
        {% endfor %}
      </tbody>
    </table>
    {% else %}
    <p>No records found matching your search criteria.</p>
    {% endif %}

    <br><a href="/admin">BACK Back to Admin</a>
  </body>
  </html>
  '''

  return render_template_string(html_template,
                                df=df,
                                total_records=total_records,
                                urgency_stats=urgency_stats,
```

```
                    avg_hb=avg_hb,
                    avg_iron=avg_iron,
                    avg_weight=avg_weight,
                    search_query=search_query)

    except FileNotFoundError:
        return " ⚠️ No historical data found. <a href='/admin'> 🔙 Back to Admin</a>"
    except Exception as e:
        return f" ⚠️ Error loading historical data: {str(e)} <a href='/admin'> 🔙 Back to Admin</a>"

@app.route("/admin/stats")
def system_stats():
    """Display comprehensive system statistics"""
    try:
        # Load all data
        current_df = pd.read_csv(CSV_FILE)
        history_df = pd.read_csv(HISTORY_FILE) if os.path.exists(HISTORY_FILE) else pd.DataFrame()
        urgency_df = pd.read_csv(URGENCY_FILE) if os.path.exists(URGENCY_FILE) else pd.DataFrame()

        # Calculate statistics
        stats = {
            'total_patients': len(current_df),
            'patients_with_complete_data': len(current_df.dropna(subset=["Hb", "Iron", "Weight",
"Blood_Group"])),
            'patients_in_urgency_window': len(urgency_df),
            'total_history_records': len(history_df),
            'blood_group_distribution': current_df['Blood_Group'].value_counts().to_dict() if
'Blood_Group' in current_df.columns else {},
            'urgency_distribution': urgency_df['Urgency'].value_counts().to_dict() if 'Urgency' in
urgency_df.columns and not urgency_df.empty else {},
            'avg_hb': current_df['Hb'].mean() if 'Hb' in current_df.columns and not
current_df['Hb'].isna().all() else 0,
```

```
        'avg_iron': current_df['Iron'].mean() if 'Iron' in current_df.columns and not current_df['Iron'].isna().all() else 0,
        'avg_weight': current_df['Weight'].mean() if 'Weight' in current_df.columns and not current_df['Weight'].isna().all() else 0,
    }

    html_template = '''
    <!doctype html>
    <html>
    <head>
      <title>System Statistics</title>
      <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .stats-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); gap: 20px; }
        .stat-card { background: #f8f9fa; padding: 20px; border-radius: 8px; border-left: 4px solid #007bff; }
        .stat-card h3 { margin-top: 0; color: #495057; }
        .big-number { font-size: 2em; font-weight: bold; color: #007bff; }
        .distribution { margin: 10px 0; }
        .distribution-item { display: flex; justify-content: space-between; padding: 5px 0; border-bottom: 1px solid #dee2e6; }
        .critical { color: #dc3545; font-weight: bold; }
        .high { color: #fd7e14; font-weight: bold; }
        .moderate { color: #ffc107; font-weight: bold; }
        .low { color: #28a745; font-weight: bold; }
      </style>
    </head>
    <body>
      <h1>📈 System Statistics Dashboard</h1>

      <div class="stats-grid">
        <div class="stat-card">
```

```
    <h3>👥 Total Patients</h3>
    <div class="big-number">{{ stats.total_patients }}</div>
    <p>Registered in system</p>
  </div>

  <div class="stat-card">
    <h3>✅ Complete Medical Records</h3>
    <div class="big-number">{{ stats.patients_with_complete_data }}</div>
    <p>Patients with full medical data for ML training</p>
  </div>

  <div class="stat-card">
    <h3>🚨 In Urgency Window</h3>
    <div class="big-number">{{ stats.patients_in_urgency_window }}</div>
    <p>Patients requiring attention (next 5 days)</p>
  </div>

  <div class="stat-card">
    <h3>📚 Historical Records</h3>
    <div class="big-number">{{ stats.total_history_records }}</div>
    <p>Training data records for ML model</p>
  </div>

  <div class="stat-card">
    <h3>🩸 Average Clinical Values</h3>
    <div class="distribution">
      <div class="distribution-item">
        <span>Hemoglobin (Hb):</span>
        <strong>{{ "%.1f"|format(stats.avg_hb) }} g/dL</strong>
      </div>
      <div class="distribution-item">
```

```
            <span>Iron:</span>
            <strong>{{ "%.1f"|format(stats.avg_iron) }} µg/dL</strong>
        </div>
        <div class="distribution-item">
            <span>Weight:</span>
            <strong>{{ "%.1f"|format(stats.avg_weight) }} kg</strong>
        </div>
    </div>
</div>

<div class="stat-card">
    <h3> 🅰️ Blood Group Distribution</h3>
    <div class="distribution">
        {% for bg, count in stats.blood_group_distribution.items() %}
        <div class="distribution-item">
            <span>{{ bg }}:</span>
            <strong>{{ count }}</strong>
        </div>
        {% endfor %}
    </div>
</div>

{% if stats.urgency_distribution %}
<div class="stat-card">
    <h3> ⚠️ Current Urgency Levels</h3>
    <div class="distribution">
        {% for urgency, count in stats.urgency_distribution.items() %}
        <div class="distribution-item">
            <span class="{{ urgency.lower() }}">{{ urgency }}:</span>
            <strong>{{ count }}</strong>
        </div>
```

```
            {% endfor %}
          </div>
        </div>
        {% endif %}

        <div class="stat-card">
          <h3>🔧 System Status</h3>
          <div class="distribution">
            {% if stats.total_history_records >= 5 %}
            <div class="distribution-item">
              <span>ML Training:</span>
              <strong style="color: #28a745;">✅ Active</strong>
            </div>
            {% else %}
            <div class="distribution-item">
              <span>ML Training:</span>
              <strong style="color: #ffc107;">⚠️ Rule-based (Need more data)</strong>
            </div>
            {% endif %}

            <div class="distribution-item">
              <span>Data Completeness:</span>
              <strong>{{ "%.1f"|format((stats.patients_with_complete_data / stats.total_patients *
100) if stats.total_patients > 0 else 0) }}%</strong>
            </div>
          </div>
        </div>
      </div>

      <br><br>
```

```
        <a href="/admin" style="text-decoration: none; background: #007bff; color: white; padding: 10px 20px; border-radius: 5px;">🔙 Back to Admin</a>
    </body>
    </html>
    '''

    return render_template_string(html_template, stats=stats)

  except Exception as e:
    return f"⚠️ Error loading statistics: {str(e)} <a href='/admin'>🔙 Back to Admin</a>"

# ---------------------- GOOGLE AUTH ----------------------

GOOGLE_CLIENT_ID = "your_client_id_here"
GOOGLE_CLIENT_SECRET = "your_client_secret_here"

google_bp = make_google_blueprint(client_id=GOOGLE_CLIENT_ID,
                                  client_secret=GOOGLE_CLIENT_SECRET,
                                  scope=["profile", "email"],
                                  redirect_to="google_logged_in")
app.register_blueprint(google_bp, url_prefix="/login")

@app.route("/user")
def user_home():
  return render_template_string('''
    <h1>🩸 User Portal</h1>
    <a href="/user/signup">📝 Sign-Up</a><br><br>
    <a href="/user/signin">🔓 Sign-In</a><br><br>
    <a href="/login/google">🔐 Google Sign-In</a>
  ''')
```

```
@app.route("/user/signup", methods=["GET", "POST"])
def user_signup():
    if request.method == "POST":
        aadhaar = request.form["aadhaar"]
        phone = request.form["phone"]

        # Check if patient already exists
        exists, existing_patient = check_patient_exists(aadhaar=aadhaar, phone=phone)
        if exists:
            return f"❌ Error: User with Aadhaar {aadhaar} or Phone {phone} already registered! <a
href='/user/signin'>Sign In Instead</a> | <a href='/user/signup'>Back</a>"

        data = {
            "Name": request.form["name"],
            "DOB": request.form["dob"],
            "Phone": phone,
            "Aadhaar": aadhaar,
            "State": request.form["state"],
            "City": request.form["city"],
            "Last_Transfusion": "",
            "Next_Transfusion": "",
            "Hb": "",
            "Iron": "",
            "Weight": "",
            "Blood_Group": request.form["blood_group"],
            "Timestamp": datetime.now(),
            "Email": ""
        }

        # Add to main CSV only (not history during signup)
        df = pd.read_csv(CSV_FILE)
```

```
        df = pd.concat([df, pd.DataFrame([data])], ignore_index=True)
        df.to_csv(CSV_FILE, index=False)

        return f"✅ Registration successful! Welcome {request.form['name']}! <a
href='/user/signin'>Sign In Now</a>"

    return render_template_string('''
        <h2>👤 Sign-Up</h2>
        <form method="POST">
            Name: <input name="name" required><br><br>
            DOB: <input name="dob" type="date" required><br><br>
            Phone: <input name="phone" pattern="\\d{10}" title="Enter 10-digit phone number"
required><br><br>
            Aadhaar: <input name="aadhaar" pattern="\\d{12}" title="Enter 12-digit Aadhaar number"
required><br><br>
            State: <input name="state" required><br><br>
            City: <input name="city" required><br><br>
            Blood Group: <select name="blood_group" required>
                <option value="">Select Blood Group</option>
                {% for bg in blood_groups %}
                    <option value="{{ bg }}">{{ bg }}</option>
                {% endfor %}
            </select><br><br>
            <input type="submit" value="✅ Register">
        </form>
        <br><p>Already registered? <a href="/user/signin">Sign In Here</a></p>
    ''', blood_groups=BLOOD_GROUPS)

@app.route("/user/signin", methods=["GET", "POST"])
def user_signin():
    if request.method == "POST":
        aadhaar = request.form["aadhaar"]
```

```
        # Check if patient exists in thalassemia_patients.csv
        exists, user_data = check_patient_exists(aadhaar=aadhaar)

        if exists:
            return render_user_dashboard(user_data)
        else:
            return f'''
            <h2>❌ Sign-In Failed</h2>
            <p>Aadhaar number <strong>{aadhaar}</strong> not found in our records.</p>
            <p><a href="/user/signup">📝 Register New Account</a></p>
            <p><a href="/user/signin">🔄 Try Again</a></p>
            <p><a href="/user">🔙 Back to User Portal</a></p>
            '''

    return render_template_string('''
        <h2>🔓 Sign-In</h2>
        <form method="POST">
            <label for="aadhaar">Aadhaar Number:</label><br>
            <input name="aadhaar" pattern="\\d{12}" title="Enter 12-digit Aadhaar number"
placeholder="Enter your 12-digit Aadhaar" required><br><br>
            <input type="submit" value="➡️ Login">
        </form>
        <br><p>New user? <a href="/user/signup">Register Here</a></p>
        <p><a href="/user">🔙 Back to User Portal</a></p>
    ''')

@app.route("/google_logged_in")
def google_logged_in():
    if not google.authorized:
        return redirect(url_for("google.login"))
```

```
    resp = google.get("/oauth2/v2/userinfo")
    if not resp.ok:
        return "❌ Failed to fetch Google profile."
    info = resp.json()
    email = info["email"]
    name = info.get("name", "")

    df = pd.read_csv(CSV_FILE)
    match = df[df["Email"] == email]
    if not match.empty:
        user = match.iloc[-1]
    else:
        new_user = {
            "Name": name,
            "DOB": "", "Phone": "", "Aadhaar": "", "State": "", "City": "",
            "Last_Transfusion": "", "Next_Transfusion": "", "Hb": "", "Iron": "",
            "Weight": "", "Blood_Group": "", "Timestamp": datetime.now(), "Email": email
        }
        df = pd.concat([df, pd.DataFrame([new_user])], ignore_index=True)
        df.to_csv(CSV_FILE, index=False)
        user = new_user

    return render_user_dashboard(user)

def render_user_dashboard(user):
    return render_template_string('''
        <h2>👋 Welcome, {{ user['Name'] }}</h2>
        <p><strong>Email:</strong> {{ user['Email'] }}</p>
        <p><strong>Phone:</strong> {{ user['Phone'] }}</p>
        <p><strong>City:</strong> {{ user['City'] }}</p>
        <p><strong>Blood Group:</strong> {{ user['Blood_Group'] }}</p>
```

```
        <p><strong>Last Hb Level:</strong> {{ user.get('Hb', 'Not recorded') }}</p>
        <p><strong>Next Transfusion:</strong> {{ user.get('Next_Transfusion', 'Not scheduled') }}</p>
        <hr>
        <h3>📸 Upload Prescription Image</h3>
        <form method="POST" action="/user/upload" enctype="multipart/form-data">
            <input type="hidden" name="aadhaar" value="{{ user['Aadhaar'] }}">
            <input type="file" name="image" accept="image/*" required><br><br>
            <input type="submit" value="📤 Upload & Extract">
        </form>
        <br><a href="/user">🔙 Back</a>
    ''', user=user)

@app.route("/user/upload", methods=["POST"])
def upload_img_extract():
    try:
        aadhaar = request.form.get("aadhaar", "").strip()
        file = request.files.get("image")
        if not file or file.filename == "":
            return "❌ No file uploaded. <a href='/user'>Back</a>"

        # Save uploaded image
        ext = os.path.splitext(file.filename)[-1].lower()
        filename = f"user_{aadhaar}_{datetime.now().strftime('%Y%m%d%H%M%S')}{ext}"
        filepath = os.path.join("static", filename)
        file.save(filepath)

        # OCR
        image = Image.open(filepath)
        text = pytesseract.image_to_string(image).lower()

        print("------ OCR EXTRACTED TEXT ------")
```

```
print(text)
print("--------------------------------")

import re

def extract_value(label, date=False):
    pattern = r''
    if date:
        # Match date like 24-07-2025 or 24/07/25
        pattern = rf"{label}[^0-9]*(\d{{1,2}}[-/]\d{{1,2}}[-/]\d{{2,4}})"
    else:
        # Match numeric values
        pattern = rf"{label}[^0-9]*([\d\.]+)"
    match = re.search(pattern, text)
    if match:
        value = match.group(1)
        if date:
            try:
                dt = datetime.strptime(value.replace("/", "-"), "%d-%m-%Y")
                return dt.strftime("%Y-%m-%d")
            except:
                try:
                    dt = datetime.strptime(value.replace("/", "-"), "%d-%m-%y")
                    return dt.strftime("%Y-%m-%d")
                except:
                    return ""
        else:
            try:
                return float(value)
            except:
                return ""
```

```
    return ""

# Extract values with fallbacks
hb = extract_value("hb") or extract_value("hemoglobin")
iron = extract_value("iron")
weight = extract_value("weight") or extract_value("wt")
last_transfusion = extract_value("last", date=True)
next_transfusion = extract_value("next", date=True)

# Update CSV - Override existing patient data
df = pd.read_csv(CSV_FILE)
idx = df[df["Aadhaar"].astype(str) == aadhaar].index

if not idx.empty:
    i = idx[-1]  # Get the latest record

    # Store original data for history
    original_data = df.loc[i].to_dict()

    # Update with new values
    if hb: df.loc[i, "Hb"] = hb
    if iron: df.loc[i, "Iron"] = iron
    if weight: df.loc[i, "Weight"] = weight
    if last_transfusion: df.loc[i, "Last_Transfusion"] = last_transfusion
    if next_transfusion: df.loc[i, "Next_Transfusion"] = next_transfusion

    df.loc[i, "Timestamp"] = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    df.to_csv(CSV_FILE, index=False)

    # Get updated data
    updated_data = df.loc[i].to_dict()
```

```
        # Check if patient has complete medical vitals
        has_complete_vitals = all([
            updated_data.get("Hb"), updated_data.get("Iron"), updated_data.get("Weight"),
            updated_data.get("Blood_Group"), updated_data.get("Next_Transfusion"),
            updated_data.get("Last_Transfusion")
        ])

        if has_complete_vitals:
            # Calculate urgency
            score, urgency_text = calculate_urgency_level(updated_data)
            # Add to history
            add_to_history(updated_data, urgency_text)
            # Trigger ML training and prediction
            train_and_predict_urgency(patient_aadhaar=aadhaar)
        else:
            print(" ⚠️ Patient updated but missing complete medical vitals for ML training")

        return render_template_string('''
            <h2>✅ Extraction & Update Successful</h2>
            <p><strong>Hb:</strong> {{ hb }}</p>
            <p><strong>Iron:</strong> {{ iron }}</p>
            <p><strong>Weight:</strong> {{ weight }}</p>
            <p><strong>Last Transfusion:</strong> {{ last }}</p>
            <p><strong>Next Transfusion:</strong> {{ nxt }}</p>
            <p><strong>Uploaded Image:</strong></p>
            <img src="/static/{{ image_path }}" style="max-width: 400px; border: 2px solid
gray;"><br><br>
            <a href="/user">🔙 Back</a>
        ''', hb=hb, iron=iron, weight=weight, last=last_transfusion,
             nxt=next_transfusion, image_path=filename)
```

```
        else:
            return "❌ Aadhaar not found in records. <a href='/user'>Back</a>"

    except Exception as e:
        return f"❌ Internal Error: {str(e)} <a href='/user'>Back</a>"

# Additional utility routes
@app.route("/")
def home():
    return render_template_string('''
        <h1>🩸 Thalassemia Management System</h1>
        <div style="margin: 20px 0;">
            <h3>Choose Portal:</h3>
            <a href="/admin" style="display: inline-block; margin: 10px; padding: 15px 25px; background-color: #007bff; color: white; text-decoration: none; border-radius: 5px;">👨‍⚕️ Admin Portal</a>
            <a href="/user" style="display: inline-block; margin: 10px; padding: 15px 25px; background-color: #28a745; color: white; text-decoration: none; border-radius: 5px;">👤 User Portal</a>
        </div>
        <div style="margin-top: 30px; padding: 20px; background-color: #f8f9fa; border-radius: 5px;">
            <h4>System Features:</h4>
            <ul>
                <li>🏥 Admin can register patients with complete medical records</li>
                <li>👤 Users can self-register and upload prescription images</li>
                <li>📊 ML-powered urgency prediction using multiple algorithms</li>
                <li>🚨 Real-time urgency monitoring and alerts</li>
                <li>📈 Historical data tracking for better predictions</li>
                <li>🔄 Automatic urgency table updates with duplicate removal</li>
                <li>📋 Comprehensive admin dashboard with statistics</li>
            </ul>
        </div>
```

```
    ''')

@app.route("/api/trigger_urgency_update", methods=["POST"])
def trigger_urgency_update():
    """API endpoint to manually trigger urgency prediction update"""
    try:
        train_and_predict_urgency()
        return jsonify({"status": "success", "message": "Urgency predictions updated successfully"}), 200
    except Exception as e:
        return jsonify({"status": "error", "message": str(e)}), 500

# Background task to periodically update urgency predictions
def periodic_urgency_update():
    """Run urgency update every hour"""
    import time
    while True:
        try:
            time.sleep(3600)  # Wait 1 hour
            print(" 🔄 Running periodic urgency update...")
            train_and_predict_urgency()
        except Exception as e:
            print(f" ⚠️ Error in periodic update: {e}")

# Start background task
def start_background_tasks():
    update_thread = Thread(target=periodic_urgency_update, daemon=True)
    update_thread.start()

# Run Flask app
def run_flask():
    start_background_tasks()
```

```
    app.run(debug=False, use_reloader=False, host='0.0.0.0', port=5000)


if __name__ == "__main__":
    # Initial urgency calculation on startup
    print("🚀 Starting Thalassemia Management System...")
    print("📊 Running initial urgency predictions...")
    train_and_predict_urgency()

    # Start the Flask application
    Thread(target=run_flask).start()
```