# AI-Powered Medical Report Simplifier

This document provides a complete guide to the AI-Powered Medical Report Simplifier, a backend service built with Flask. It takes medical reports (as text or images) and produces patient-friendly explanations.

## Table of Contents

1. [Project Structure](https://www.google.com/search?q=%23project-structure)
2. [Dependencies (requirements.txt)](https://www.google.com/search?q=%23dependencies-requirementstxt)
3. [Final Application Code (app.py)](https://www.google.com/search?q=%23final-application-code-apppy)
4. [Complete Setup Guide](https://www.google.com/search?q=%23complete-setup-guide)
5. [Running the Application](https://www.google.com/search?q=%23running-the-application)
6. [Testing with Postman](https://www.google.com/search?q=%23testing-with-postman)
7. [Exposing Your Server with Ngrok (for Windows / Git Bash)](https://www.google.com/search?q=%23exposing-your-server-with-ngrok-for-windows--git-bash)
8. [Understanding the "404 Not Found" Error](https://www.google.com/search?q=%23understanding-the-404-not-found-error)
9. [How to Stand Out: Project Enhancements](https://www.google.com/search?q=%23how-to-stand-out-project-enhancements)

## 1. Project Structure

For this project, your folder in VS Code should be organized as follows:

Plum-assignment/  
├── venv/ # Python virtual environment  
├── app.py # The main Flask application script  
├── requirements.txt # Lists the Python packages needed  
└── README.md # This guide

## 2. Dependencies (requirements.txt)

Create a file named requirements.txt and paste the following content into it. This ensures you have the correct, easy-to-install libraries.

Flask>=3.0.0  
Pillow>=10.0.0  
pytesseract  
rapidfuzz

## 3. Final Application Code (app.py)

Create a file named app.py and paste the entire code block below into it.

import os  
import re  
import io  
import json  
from flask import Flask, request, jsonify  
from PIL import Image  
import pytesseract  
# Use rapidfuzz for fast and easy-to-install string matching  
from rapidfuzz import process, fuzz  
  
# --- Configuration ---  
# If Tesseract is not in your system's PATH, you may need to set its location.  
# For Windows, the path might look like this:  
# pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'  
  
app = Flask(\_\_name\_\_)  
  
# --- Mock Database for Test Normalization ---  
# In a real-world scenario, this would be a comprehensive, external database.  
KNOWN\_TESTS = {  
 "Hemoglobin": {"unit": "g/dL", "ref\_range": {"low": 12.0, "high": 15.0}},  
 "WBC": {"unit": "/uL", "ref\_range": {"low": 4000, "high": 11000}},  
 "White Blood Cell Count": {"unit": "/uL", "ref\_range": {"low": 4000, "high": 11000}},  
 "Platelets": {"unit": "10^3/uL", "ref\_range": {"low": 150, "high": 450}},  
 "Glucose": {"unit": "mg/dL", "ref\_range": {"low": 70, "high": 100}},  
}  
  
# --- 1. OCR and Text Extraction ---  
def extract\_text\_from\_image(image\_file):  
 """Performs OCR on an image file to extract text using Tesseract."""  
 try:  
 image = Image.open(io.BytesIO(image\_file.read()))  
 text = pytesseract.image\_to\_string(image)  
 return text, 0.95 # Assume a high confidence for any successful OCR  
 except Exception as e:  
 print(f"OCR Error: {e}")  
 return None, 0.0  
  
def extract\_raw\_tests\_from\_text(text):  
 """Extracts individual test lines from raw text using simple heuristics."""  
 lines = text.strip().split('\n')  
 test\_lines = [line.strip() for line in lines if re.search(r'\d', line)]  
 confidence = min(1.0, len(test\_lines) / 5.0)  
 return {"tests\_raw": test\_lines, "confidence": confidence}  
  
# --- 2. Test Normalization ---  
def normalize\_tests(raw\_tests):  
 """Normalizes raw test strings into a structured JSON format using rapidfuzz."""  
 normalized = []  
 total\_confidence = 0  
 for raw\_test in raw\_tests:  
 try:  
 test\_name\_match = process.extractOne(raw\_test, KNOWN\_TESTS.keys(), scorer=fuzz.WRatio)  
 if not test\_name\_match or test\_name\_match[1] < 70:  
 continue  
 name = test\_name\_match[0]  
 total\_confidence += test\_name\_match[1] / 100.0  
 value\_match = re.search(r'(\d[\d,]\*\.?\d\*)', raw\_test)  
 if not value\_match:  
 continue  
 value = float(value\_match.group(1).replace(',', ''))  
 ref\_range = KNOWN\_TESTS[name]['ref\_range']  
 status = "normal"  
 if value < ref\_range['low']:  
 status = "low"  
 elif value > ref\_range['high']:  
 status = "high"  
 normalized.append({  
 "name": name, "value": value, "unit": KNOWN\_TESTS[name]['unit'],  
 "status": status, "ref\_range": ref\_range  
 })  
 except Exception as e:  
 print(f"Error normalizing test: '{raw\_test}'. Error: {e}")  
 continue  
 avg\_confidence = (total\_confidence / len(raw\_tests)) if raw\_tests else 0  
 return {"tests": normalized, "normalization\_confidence": round(avg\_confidence, 2)}  
  
# --- 3. Patient-Friendly Summary Generation ---  
def generate\_summary(normalized\_tests):  
 """Generates a patient-friendly summary from normalized test results."""  
 abnormal\_tests = [t for t in normalized\_tests if t['status'] != 'normal']  
 if not abnormal\_tests:  
 return {"summary": "All test results appear to be within the normal range.", "explanations": [], "status": "ok"}  
   
 summary\_parts, explanations = [], []  
 for test in abnormal\_tests:  
 summary\_parts.append(f"{test['status']} {test['name'].lower()}")  
 if test['name'] == 'Hemoglobin' and test['status'] == 'low':  
 explanations.append("Low hemoglobin might be related to a condition called anemia, where your body has fewer red blood cells than normal. This can cause feelings of tiredness or weakness.")  
 elif (test['name'] == 'WBC' or test['name'] == 'White Blood Cell Count') and test['status'] == 'high':  
 explanations.append("High white blood cells can be a sign that your body is fighting an infection.")  
 summary = "Your results show " + " and ".join(summary\_parts) + "."  
   
 original\_abnormal\_names = {t['name'].lower() for t in abnormal\_tests}  
 explanation\_text = " ".join(explanations).lower()  
 for known\_test in KNOWN\_TESTS:  
 if known\_test.lower() in explanation\_text and known\_test.lower() not in original\_abnormal\_names:  
 return {"status": "unprocessed", "reason": f"potential hallucinated test '{known\_test}' in explanation"}  
 return {"summary": summary, "explanations": explanations, "status": "ok"}  
  
# --- 4. API Endpoint ---  
@app.route('/simplify', methods=['POST'])  
def simplify\_report():  
 """Main API endpoint to process medical reports from text or images."""  
 if 'image' in request.files:  
 image\_file = request.files['image']  
 text, \_ = extract\_text\_from\_image(image\_file)  
 if not text:  
 return jsonify({"error": "Could not extract text from image."}), 400  
 elif 'text' in request.form:  
 text = request.form['text']  
 else:  
 return jsonify({"error": "Request must contain either 'text' or 'image' data."}), 400  
  
 raw\_results = extract\_raw\_tests\_from\_text(text)  
 if not raw\_results['tests\_raw']:  
 return jsonify({"error": "No valid test results found in the input text."}), 400  
  
 normalization\_result = normalize\_tests(raw\_results['tests\_raw'])  
 if not normalization\_result['tests']:  
 return jsonify({"status":"unprocessed", "reason":"Could not normalize any tests from the input"}), 400  
  
 summary\_result = generate\_summary(normalization\_result['tests'])  
 if summary\_result['status'] != 'ok':  
 return jsonify(summary\_result), 400  
  
 final\_output = {  
 "tests": normalization\_result['tests'],  
 "summary": summary\_result['summary'],  
 "explanations": summary\_result['explanations'],  
 "status": "ok"  
 }  
 return jsonify(final\_output)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True, port=5000)

## 4. Complete Setup Guide

### Step A: Prerequisites

1. **Python 3.8+**: Ensure you have a modern version of Python installed.
2. **Tesseract OCR Engine**: This is mandatory for processing images.
   * **Windows**: Download and run the installer from the [Tesseract at UB Mannheim](https://www.google.com/search?q=https://github.com/UB-Mannheim/tesseract/wiki) page. **Crucial:** During installation, check the box to add Tesseract to your system's PATH.
   * **macOS (Homebrew)**: brew install tesseract
   * **Linux (Ubuntu/Debian)**: sudo apt update && sudo apt install tesseract-ocr

### Step B: Project Installation

1. **Create a project folder** (e.g., Plum-assignment) and place the app.py and requirements.txt files inside it.
2. **Open a Git Bash terminal** and navigate into your project folder.
3. **Create and activate a Python virtual environment**:  
   # Create the environment  
   python -m venv venv  
     
   # Activate the environment  
   source venv/Scripts/activate
4. **Install all required packages**:  
   pip install -r requirements.txt

## 5. Running the Application

With your virtual environment still active, run the following command in your Git Bash terminal:

python app.py

The server is now running locally at http://127.0.0.1:5000.

## 6. Testing with Postman

Postman is the best way to test your API. Use the Postman Desktop application for the best experience.

### Test 1: Sending Text Data

1. Open Postman and create a new request.
2. Set the method to **POST**.
3. Set the URL to http://127.0.0.1:5000/simplify.
4. Go to the **Body** tab and select **form-data**.
5. In the KEY field, enter text. In the VALUE field, enter CBC: Hemglobin 10.2 g/dL (Low), WBC 11200 /uL (Hgh).
6. Click **Send**.

### Sample Image for Testing

To test the image upload, you can right-click and save the image below.

### Test 2: Sending an Image File

1. Use the same POST request to http://127.0.0.1:5000/simplify.
2. In the **Body** -> **form-data** tab, enter image in the KEY field.
3. Hover over the image key, click the dropdown that says "Text", and change it to **"File"**.
4. Click the "Select Files" button in the VALUE column and upload the sample image.
5. Click **Send**.

## 7. Exposing Your Server with Ngrok (for Windows / Git Bash)

ngrok creates a public URL for your local server, allowing others to test your live application.

### Step A: Setup Ngrok on Windows

1. **Download Ngrok**: Go to the [ngrok download page](https://www.google.com/search?q=https://ngrok.com/download) and get the ZIP file for Windows.
2. **Unzip the File**: This will give you a single file, ngrok.exe. Move this file to a simple, memorable location (e.g., C:\Users\YourUsername\).
3. **Sign Up & Get Authtoken**:
   * Sign up for a free account on the [ngrok website](https://dashboard.ngrok.com/signup).
   * From your dashboard, find "Your Authtoken" and copy the full ngrok config add-authtoken ... command.
4. **Connect Your Account**:
   * Open a **Git Bash terminal**.
   * Navigate to the folder where you saved ngrok.exe. For example: cd C:/Users/YourUsername/.
   * Paste and run the command from your dashboard. **Important:** You must replace <YOUR\_TOKEN> with the actual token you copied from the website.
   * Add ./ before the command to tell Git Bash to run the ngrok.exe file in the current folder.  
     Example: If your token is 2abc...xyz, the command will look like this:  
     ./ngrok config add-authtoken 2abcDEfgHiJKLmnOPqrsTUVwxYZ1234567\_SAMPLETOKEN
   * This authenticates your Windows ngrok.exe.

### Step B: Run Ngrok from Git Bash

1. **Make sure your Flask server is running** in one Git Bash terminal.
2. **Open a second Git Bash terminal**.
3. **Navigate to the folder containing ngrok.exe**.
4. **Run the following command**. This tells the ngrok.exe in the current folder to expose your Flask server.  
   ./ngrok http 5000
5. **Get Your Public URL**: The terminal will now display your public https URL (e.g., https://<random-string>.ngrok-free.app).

### Step C: Troubleshooting "Endpoint Already Online" Error

If you get an error that the endpoint is already online (ERR\_NGROK\_334), it means an old ngrok process is stuck running in the background.

* **Note for WSL Users:** If you have Windows Subsystem for Linux (WSL) and have run ngrok from there, the process might be running inside your Linux environment. The Windows taskkill command **cannot** stop it. You must first stop the Linux process. Open a WSL terminal and run ps aux | grep ngrok to find the Process ID (PID), then run kill <PID> to stop it.
* **Stopping the Windows Process:** The most reliable way to stop a stuck Windows ngrok process is to kill it by name. Run this single command in your Git Bash terminal:  
  # This command works in Git Bash to stop Windows processes by name  
  taskkill //F //IM ngrok.exe