

Medical Report Simplifier

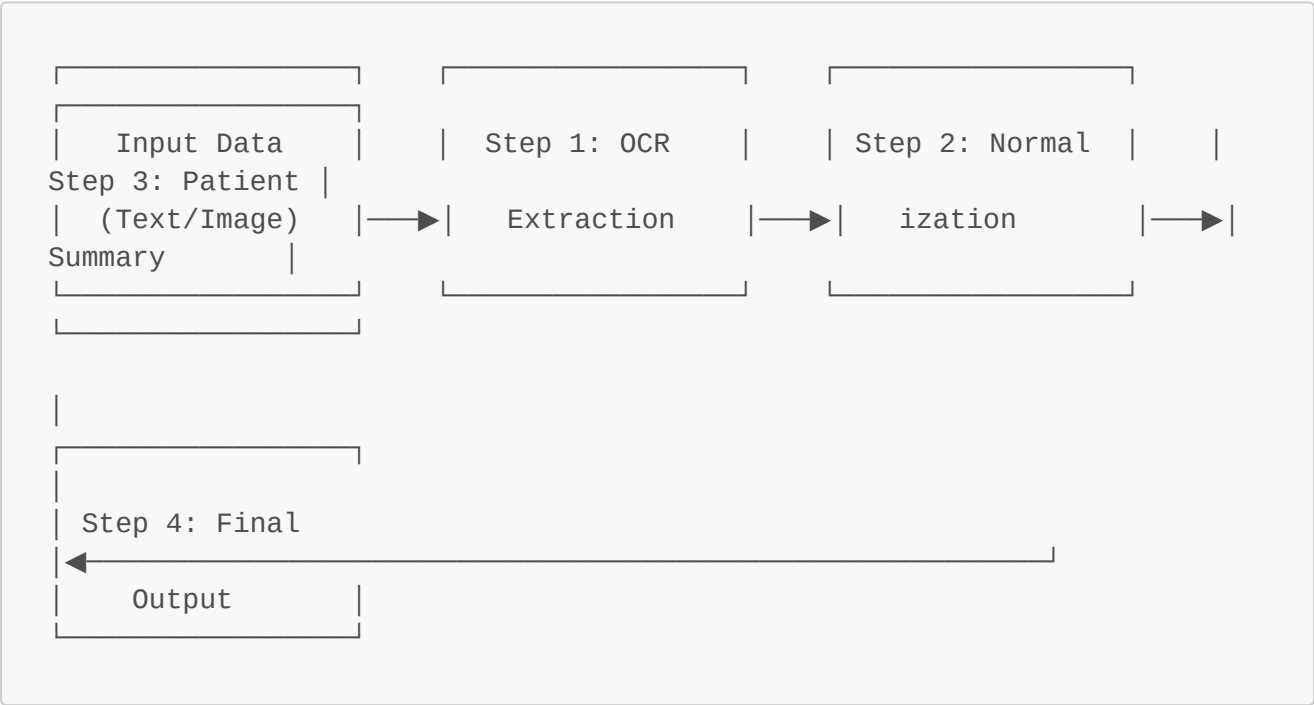
An AI-powered backend service that processes medical reports (text or images) and provides patient-friendly explanations. Built with FastAPI, Google Gemini AI, and Tesseract OCR to transform complex medical data into understandable information.

Features

- **Multi-format Input:** Process both text and images (PNG, JPG, JPEG, BMP, TIFF)
- **OCR Processing:** Tesseract-based text extraction with confidence scoring
- **AI-Powered Processing:** Google Gemini AI for error correction and normalization
- **Smart Normalization:** Standardizes medical test names, values, units, and reference ranges
- **Patient-Friendly Explanations:** Simple, non-technical summaries
- **Validation:** AI semantic validation prevents fabricated test results
- **4-Step Pipeline:** Complete processing from raw input to final output
- **Error Handling:** Proper validation with detailed error responses

Architecture

The system follows a 4-step processing pipeline:



Component Overview:

- **FastAPI Application:** RESTful API with automatic documentation
- **OCR Service:** Tesseract integration for image text extraction
- **AI Service:** Google Gemini integration for intelligent processing
- **Processing Service:** Main business logic coordinator
- **Validation:** Input validation and error handling

Setup Instructions

Prerequisites

- Python 3.9 or higher
- Tesseract OCR installed on your system
- Google Gemini API key

1. Clone the Repository

```
git clone <repository-url>
cd medical-report-simplifier
```

2. Install System Dependencies

Ubuntu/Debian:

```
sudo apt-get update
sudo apt-get install tesseract-ocr tesseract-ocr-eng
sudo apt-get install libmagic1
```

macOS:

```
brew install tesseract
brew install libmagic
```

Windows:

1. Download and install [Tesseract OCR](#)
2. Add Tesseract to your PATH

3. Install Python Dependencies

```
# Create virtual environment (recommended)
python -m venv venv
source venv/bin/activate # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

4. Environment Configuration

Create a `.env` file in the project root:

```
cp .env.example .env # If example exists, or create new file
```

5. Start the Application

```
# Development mode
python start.py

# Or using uvicorn directly
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

The API will be available at <http://localhost:8000>

Environment Configuration

Create a `.env` file with the following variables:

```
# Required
GEMINI_API_KEY=your_google_gemini_api_key_here

# Optional (with defaults)
DEBUG=True
LOG_LEVEL=INFO
MAX_FILE_SIZE=10485760 # 10MB in bytes
PORT=8000
TESSERACT_PATH=/usr/bin/tesseract # Adjust path as needed
```

—
PROF

Getting Google Gemini API Key:

1. Visit [Google AI Studio](#)
2. Create a new API key
3. Add it to your `.env` file

API Endpoints

Health Check

- **GET** [/api/v1/health](#) - Check service health and configuration

Production Endpoints

- **POST** [/api/v1/process-text](#) - Process medical report text
- **POST** [/api/v1/process-image](#) - Process medical report image

Demo Endpoint

- **POST** `/api/v1/demo-problem-statement` - Returns 4-step processing breakdown

Documentation

- **GET** `/docs` - Interactive Swagger UI documentation
- **GET** `/redoc` - ReDoc documentation
- **GET** `/` - API information and links

Usage Examples

Processing Text Input

```
import requests
import json

url = "http://localhost:8000/api/v1/process-text"
data = {
    "text": "Blood Test Results: Hemoglobin: 12.5 g/dL (normal: 12-16),
    Glucose: 180 mg/dL (normal: 70-100)"
}

response = requests.post(url, json=data)
result = response.json()
print(json.dumps(result, indent=2))
```

Processing Image Input

```
import requests

url = "http://localhost:8000/api/v1/process-image"

with open("medical_report.jpg", "rb") as f:
    files = {"file": ("medical_report.jpg", f, "image/jpeg")}
    response = requests.post(url, files=files)

result = response.json()
print(json.dumps(result, indent=2))
```

Sample Requests

1. Health Check (cURL)

```
curl -X GET "http://localhost:8000/api/v1/health" \
-H "accept: application/json"
```

Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "environment": {
    "python_version": "3.9",
    "port": "8000",
    "gemini_configured": true,
    "magic_available": true
  }
}
```

2. Process Text (cURL)

```
curl -X POST "http://localhost:8000/api/v1/process-text" \
  -H "accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "Lab Results: Hemoglobin 11.2 g/dL (Normal: 12.0-16.0),
White Blood Cell Count 8500 /uL (Normal: 4000-11000), Glucose 165 mg/dL
(Normal: 70-100)"
  }'
```

3. Process Image (cURL)

```
curl -X POST "http://localhost:8000/api/v1/process-image" \
  -H "accept: application/json" \
  -F "file=@/path/to/medical_report.jpg"
```

4. Demo Problem Statement Format (cURL)

```
curl -X POST "http://localhost:8000/api/v1/demo-problem-statement" \
  -H "accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{
    "text": "CBC Report: RBC 4.2 million/uL, WBC 7800/uL, Platelets
250000/uL, Hemoglobin 13.5 g/dL"
  }'
```

Testing

Manual Testing Examples

1. Basic Health Check

```
curl -X GET "http://localhost:8000/api/v1/health"
```

2. Simple Text Processing

```
curl -X POST "http://localhost:8000/api/v1/process-text" \
-H "Content-Type: application/json" \
-d '{"text": "CBC Results: Hemoglobin 9.5 g/dL (Low), WBC 12000/uL (High)"}'
```

3. Demo 4-Step Format

```
curl -XPOST "http://localhost:8000/api/v1/demo-problem-statement" \
-H "Content-Type: application/json" \
-d '{"text": "CBC: RBC 4.2 million/uL, WBC 7800/uL, Hemoglobin 13.5 g/dL"}'
```

4. Error Handling Test

```
curl -X POST "http://localhost:8000/api/v1/process-text" \
-H "Content-Type: application/json" \
-d '{"text": "This is not medical data"}'
```

PROF

Technology Stack

Component	Technology	Purpose
Web Framework	FastAPI 0.104+	REST API with automatic documentation
AI/ML	Google Gemini AI	Text processing, normalization, summarization
OCR Engine	Tesseract OCR	Image text extraction
Data Validation	Pydantic 2.5+	Request/response validation
Image Processing	Pillow (PIL)	Image format handling
File Handling	python-multipart	File upload support
HTTP Client	httpx	API client for testing

Component	Technology	Purpose
Testing	pytest	Unit and integration tests
Environment	python-dotenv	Configuration management

Project Structure

```

medical-report-simplifier/
├── app/                                # Main application package
│   ├── main.py                        # FastAPI app entry point
│   ├── api/
│   │   └── endpoints.py               # API route definitions
│   ├── core/
│   │   └── 📄 config.py                # Configuration settings
│   ├── models/
│   │   └── schemas.py                 # Pydantic models
│   └── services/
│       ├── ai_service.py              # Gemini AI integration
│       ├── ocr_service.py             # OCR processing
│       └── processing_service.py      # Main processing pipeline
├── requirements.txt                    # Python dependencies
├── Dockerfile                         # Docker configuration
├── start.py                           # Development server starter
└── README.md                          # This documentation

```