Medical Diagnostic Assistant - Technical Documentation

Overview

The Medical Diagnostic Assistant is an AI-powered medical diagnosis system that conducts virtual consultations through a question-and-answer format. It leverages Google's Gemini 1.5 Flash model to analyze patient symptoms, ask appropriate follow-up questions, and provide diagnostic suggestions based on the conversation.

System Architecture

Components

1. **Core Diagnostic Engine (`prototype1.py`)**:
    - Handles the Gemini API integration
    - Implements the diagnostic conversation flow
    - Manages question selection and categorization
    - Generates medical reports
2. **Flask API Server (`app1.py`)**:
    - Provides HTTP endpoints for the diagnostic system
    - Maintains conversation state
    - Communicates with external storage APIs

Data Flow

1. Patient submits initial symptoms
2. System categorizes symptoms (Heart failure, Tuberculosis, General consultation)
3. AI assistant generates appropriate follow-up questions
4. Responses are processed and stored
5. After sufficient information is gathered, diagnostic reports are generated

Key Features

Symptom Classification

The system classifies conversations into medical categories to guide the question-selection process:

- Heart failure
- Tuberculosis
- General consultation

Intelligent Question Selection

- Pulls from a predefined question bank based on symptom category
- Avoids repetitive questions
- Dynamically selects next best question based on previous responses

Dual Report Generation

- **Doctor Report**: Technical medical analysis with differential diagnosis and next steps.
- **Patient Report**: Simplified explanation of condition and next steps

External Data Storage

- Conversations and reports are stored both locally and in a remote API
- Each consultation has a unique conversation ID
- Implementation includes authentication via API tokens

Technical Specifications

Dependencies

- Python 3.x
- Flask (web framework)
- Requests (HTTP library)
- dotenv (environment configuration)

API Endpoints

`/ai_conversation` (POST)

- **Headers**:

    - `x-api-key`: API authentication key
    - `authorization`: User authorization token
    - `user-id`: Unique identifier for the patient

Request Body:

```
{
 "conversation_id": "[ unique identifier]",
 "user_response": "[patient's answer]",
 "question": "[previous question]"
}
```

Response:

```
{
 "flag": "[question|diagnosis_complete|fallback]",
 "Q": "[next question]",
 "D": "[diagnosis if complete]",
 "patient_report": "[simplified patient report]",
 "conversation_logs": "[history of the conversation]",
 "question_count": "[number of questions asked]"
}
```

Environment Variables

- `API_KEY`: Google Gemini API key
- `X_API_KEY`: Service authentication key

Authentication

Two-layer authentication:

1. API key validation for service access
2. User authorization token for conversation persistence

Implementation Details

Conversation Flow Logic

1. System starts with initial symptoms assessment

2. Requires minimum 5 questions before diagnosis
3. Limits to maximum 15 questions per consultation
4. Dynamically evaluates when sufficient information is gathered

AI Query Construction

- Context-aware prompts include:
    - Current symptoms
    - Conversation history
    - Available questions
    - Response format instructions

Report Generation

- Doctor reports include:
    - Whom to approach (specialist recommendation)
    - Differential diagnosis
    - Suggested next steps
- Patient reports include:
    - Simplified condition summary
    - Recommended next steps

Error Handling

- Fallback questions for unexpected API responses
- JSON parsing error recovery
- API timeout handling

Release Note: Medical Diagnostic Assistant v3.0

Release Date: April 24, 2025

Announcing the release of our Medical Diagnostic Assistant v3.0, a sophisticated AI-powered system designed to conduct virtual medical consultations and provide diagnostic guidance.

What's New

- **AI-Powered Diagnostic Engine**: Our system leverages Google's Gemini 1.5 Flash model to intelligently analyze symptoms and guide patients through a medical consultation.
- **Category-Based Questioning**: The system identifies symptom categories and selects follow-up questions from a curated database.
- **Dual Reporting System**: Generates both technical reports for healthcare providers and simplified explanations for patients.
- **REST API**: Easy integration with existing healthcare applications through our documented API.

Key Features

- Intelligent follow-up question selection
- Symptom classification into medical categories
- Progressive diagnostic assessment
- Secure conversation storage and retrieval
- Customizable minimum and maximum question thresholds

Technical Highlights

- Python-based implementation with Flask web framework
- Token-based authentication system
- External API integration for data persistence
- Local and remote storage options for consultation records

Getting Started

1. Set up the required environment variables:
    - `API_KEY` for Gemini API access
    - `X_API_KEY` for service authentication

Run the application:

```
python app1.py
```

2. Access the API endpoint at `http://localhost:5000/ai_conversation`

Notes for Developers

- The system requires valid authorization tokens for conversation persistence
- The project includes both CLI-based testing (`prototype1.py`) and API-based deployment (`app1.py`)
- Consultation data is stored both locally and in the remote API