Real-Time AI-Powered Symptom Checker and Specialist Recommender Using Gemini API

In real-world healthcare systems, users often search for symptom explanations and appropriate specialists by entering vague or unstructured queries on the web. The challenge is to interpret natural language health descriptions and provide reliable, intelligent suggestions **without relying on static medical datasets**.

As data engineers of the future, trainees are tasked with building **an intelligent, scalable, and real-time system** that harnesses the capabilities of **large language models (LLMs)** to bridge the gap between symptom reporting and professional medical guidance.

Objective:

Design and implement a **real-time intelligent healthcare assistant** that:

1. Accepts natural language symptom descriptions from a user via a **Streamlit web app**.
2. Uses **Gemini API** to:
    - Interpret and summarize the reported symptoms.
    - Identify **likely medical conditions**.
    - Recommend the **type of specialist** best suited for the patient.
3. Presents the output in a clean, interactive interface **with no reliance on static data files or mock APIs**.

This project should **mimic a real-world AI system** capable of delivering contextualized, LLM-driven outputs using a single intelligent service (Gemini API), demonstrating how **data engineering is evolving beyond traditional pipelines into live inference and API-based architectures.**

Core Tasks:

- Create a **Streamlit interface** for symptom input and results display.
- Design a clean, modular **Python backend** for handling user input and communicating with the Gemini API.
- Build a **prompting engine** that effectively transforms user input into structured Gemini API prompts.
- Handle response parsing, error handling, and edge case management in real-time.

Constraints:

- **No static datasets**, mock data, or offline condition mappings are allowed.
- All intelligence must come from **Gemini API in real time**.
- The focus should be on **intelligent pipeline orchestration**, not UI design or storage.

Expected Deliverables:

1. Complete project code including:
    - Streamlit app (main.py)
    - Gemini API handler module (gemini_api.py)
2. Clear documentation (README.md) explaining:
    - Architecture
    - Prompting strategy
    - Setup and run instructions
3. Optional: Add **logging** and **prompt response analysis** to evaluate Gemini API effectiveness.

Tech Stack:

- **Python** (PyCharm IDE)
- **Streamlit** (Web UI)
- **Gemini API** (NLP and medical logic)
- Optional: **LangChain**, **logging**, or **prompt engineering utilities**