**The Future of Diagnostic Solutions with Gemini**

A revolutionary generative AI project designed to assist healthcare professionals in the diagnosis and treatment of patients. This industry-ready platform leverages advanced machine learning algorithms trained on vast medical datasets to provide real-time insights and improve clinical decision-making.

**Scenario 1: Enhanced Diagnostic Support**

Physicians often face challenges in diagnosing complex medical conditions, particularly when dealing with rare diseases or overlapping symptoms. PrognosisAI addresses this by providing intelligent diagnostic support. Doctors can input patient symptoms, medical history, and test results. The AI tool then analyzes this data alongside its extensive medical knowledge base to generate a differential diagnosis list, prioritizing the most likely conditions based on the available information. This empowers physicians to make more informed diagnoses and expedite the treatment process.

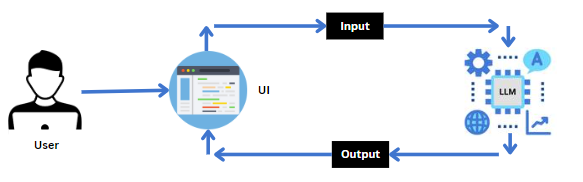
**Scenario 2: Personalized Treatment Recommendations**

Developing personalized treatment plans is crucial for optimal patient outcomes. PrognosisAI facilitates this by recommending evidence-based treatment options tailored to each patient's specific needs. By considering factors like patient demographics, medical history, and the severity of the condition, the AI suggests appropriate treatment modalities, medication dosages, and potential side effects. This empowers doctors to personalize care plans and optimize treatment efficacy.

**Scenario 3: Clinical Trial Matching**

Clinical trials play a vital role in medical advancements. However, matching patients with suitable trials can be a cumbersome process. PrognosisAI streamlines this by acting as a clinical trial matching platform. Patients can input their medical information, and the AI tool identifies ongoing trials that align with their specific condition. This facilitates patient participation in relevant research endeavors, accelerating medical breakthroughs and offering potential treatment options not yet widely available.

**Architecture:**



**Project Flow:**

* **User Interaction:**
  + Healthcare professionals input patient data via a user-friendly web application.
  + Data includes patient symptoms, medical history, demographics, and test results.

* **Backend Processing:**
* AI Analysis:
  + AI analyzes patient data against a medical knowledge base.
* Results Generation: The AI generates differential diagnoses.
* **User Output:**
  + The results are presented to the healthcare professional via the web application UI.
  + Detailed reports and insights are provided.

To accomplish this, we have to complete all the activities listed below,

* Setting Up Google API Key
  + Generate Gemini API Key
* Installation and Importing of Libraries and Adding API Key
  + Install and Import necessary libraries for the project
  + Add the Google API Key to the code
* Define Prompt Templates
  + Create prompt template for differential diagnosis
* Collect User Inputs
  + Collect diagnostic support input from the user
* Generate AI Responses
  + Generate a differential diagnosis list
* Build Streamlit User Interface
  + Create main Streamlit application title
  + Implement diagnostic support interface

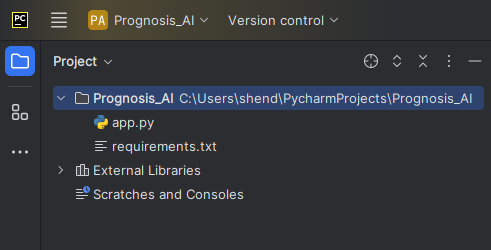
**Prior Knowledge:**

You must have prior knowledge of the following topics to complete this project.

* LLM & Google GenAI: <https://cloud.google.com/vertex-ai/docs/generative-ai/learn-resources>
* Streamlit: Create interactive web applications. https://docs.streamlit.io/
* Google Generative AI: https://ai.google.dev/gemini-api/cookbook
* Langchain: https://python.langchain.com/v0.1/docs/get\_started/introduction/

**Project Structure:**

Create the Project folder which contains files as shown below

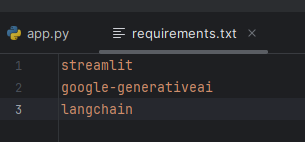


**Milestone 1: Requirements Specification**

In this milestone, we will install necessary libraries.

**Activity 1.1: Install necessary libraries for the project**

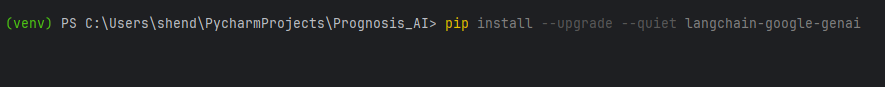
* Before installing, activate the environment for developing the project.
* Firstly install the libraries present in the requirement.txt file

****

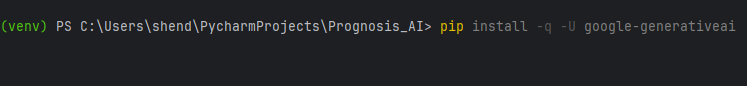
pip install -r requirements.txt

****

pip install --upgrade --quiet langchain-google-genai

****

pip install -q -U google-generativeai

****

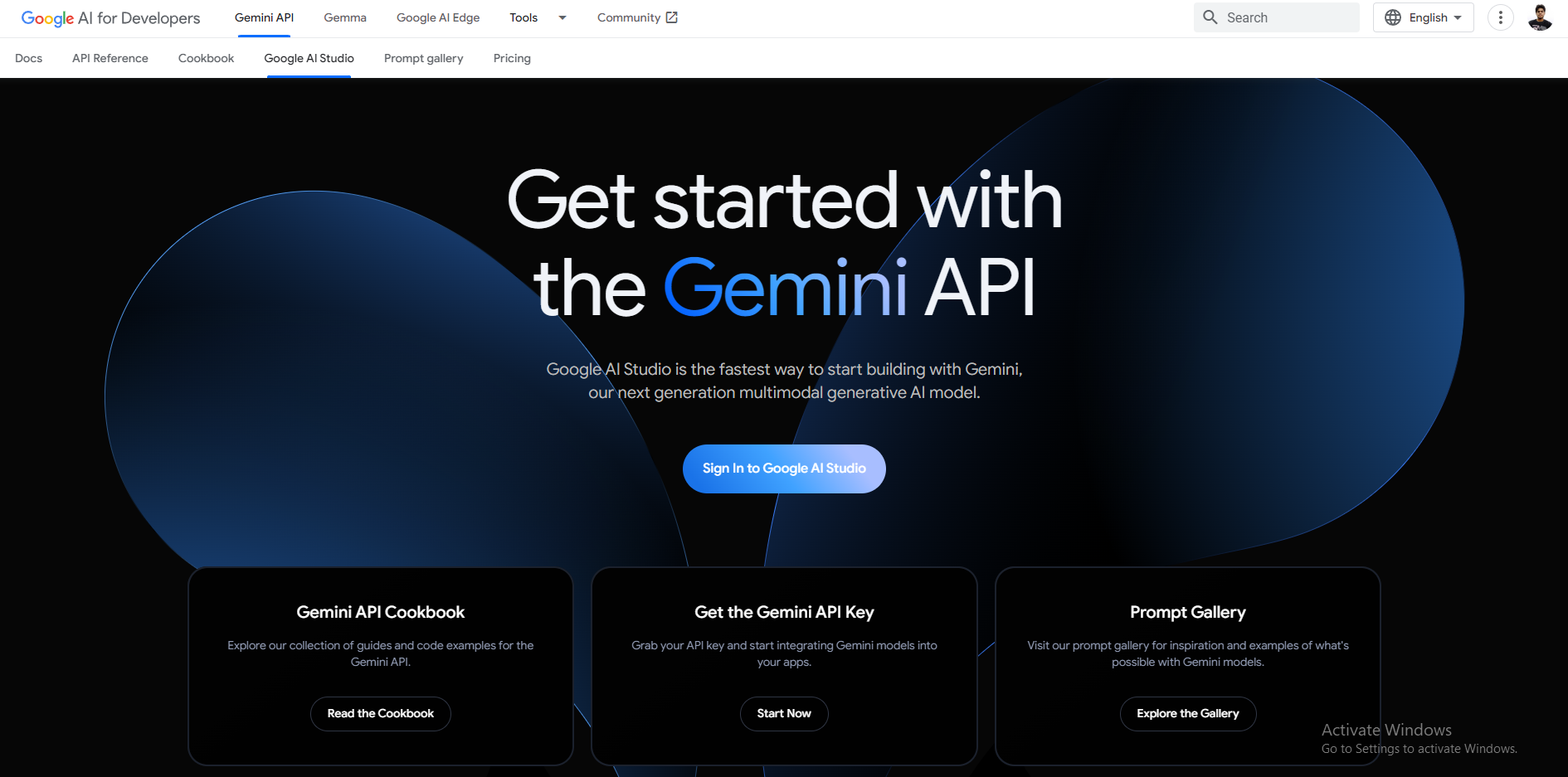
* We are building a Streamlit application in which we will use Python script app.py for scripting.
* The requirements.txt file contains all the libraries required for the project.

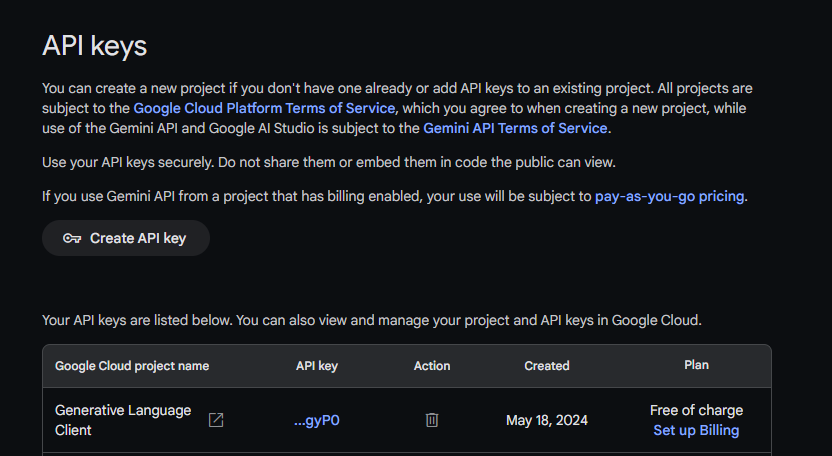
**Milestone 2: Setting Up Google API Key**

For initializing the model we need to generate PALM API.

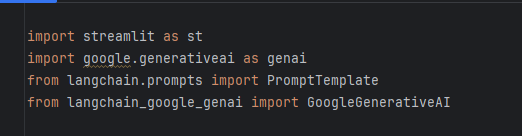
**Activity 2.1: Generate Gemini API**

* Click on the link (<https://ai.google.dev/aistudio>).
* Then click on “Get API key in Google AI Studio”.
* Click on “Get API key” from the right navigation menu.
* Now click on “Create API key”. (Refer the below images)
* Copy the API key.



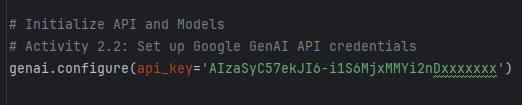


**Milestone 3: Interfacing with pre-trained Model**

****

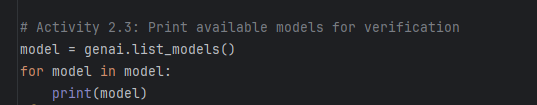
* Streamlit (streamlit): Web application framework for building interactive web apps.
* Google Generative AI (google.generativeai): Google's API for accessing generative AI models.
* langchain.prompts: This class is used to define and manage prompt templates. A prompt template allows you to define a structure for your prompts, including placeholders for dynamic content that can be filled in at runtime. This is useful for standardizing the format of the prompts you send to the language model.
* langchain\_google\_genai: This module is an integration within LangChain specifically for interfacing with Google Generative AI models. Google Generative AI refers to the AI models developed by Google for generating human-like text based on given prompts.

**Activity 3.1: Add the Google API Key to the code**

****

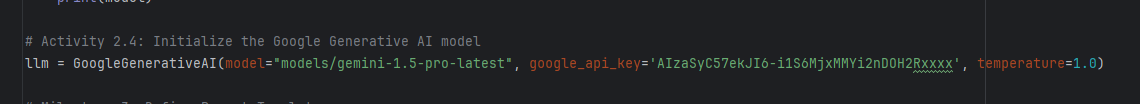
* Paste your API KEY
* The code is now configured to authenticate with Google's Generative AI services using the API key.
* Adding the API key to the code is a critical configuration step that enables the application to make authenticated requests to Google’s AI services.

**Activity 3.2: Print list of available models**

****

* Use genai.list\_models() to list available models and verify connectivity.
* Input: Google Generative AI library.
* Process: The user runs genai.list\_models() to list available models and prints them to the console.
* Output: A printed list of available models, confirming API connectivity.
* Print the models to the console to confirm availability and proper API configuration.
* Verifying available models ensures that the application can access the appropriate AI models from Google’s services. This step confirms the setup is correctly configured.

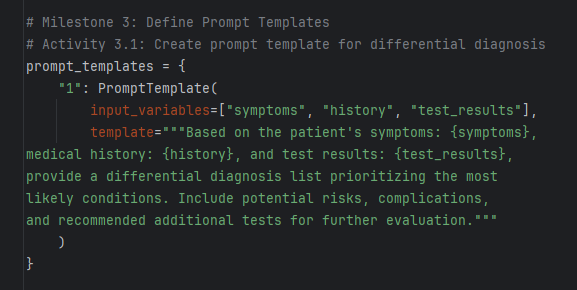
**Activity 3.3: Initialize the Google Generative AI model**



* Initialize the Google Generative AI model with desired parameters.
* Set the model temperature and API key for generating varied responses.
* Input: Model parameters and API key.
* Process: The user initializes the model using commands that set parameters like temperature and API key.
* Output: A configured and initialized generative AI model ready for use.
* Initializing the AI model with specific parameters tailors the model’s behavior to the project’s needs, setting the stage for generating diagnostic and treatment outputs.

**Milestone 4: Define Prompt Templates**

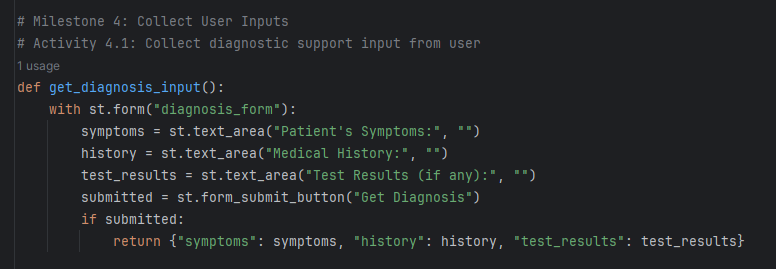
**Activity 4.1: Create prompt template for differential diagnosis**



* This activity involves defining a template for generating differential diagnosis lists.
* The template includes placeholders for patient symptoms, medical history, and test results, which will be filled in with actual data when making a request to the AI model.
* Ensure the template asks for a differential diagnosis list prioritizing likely conditions.
* Creating prompt templates standardizes how requests are made to the AI model, ensuring consistent and accurate diagnostic support based on structured patient data.

**Milestone 5: Collect User Inputs**

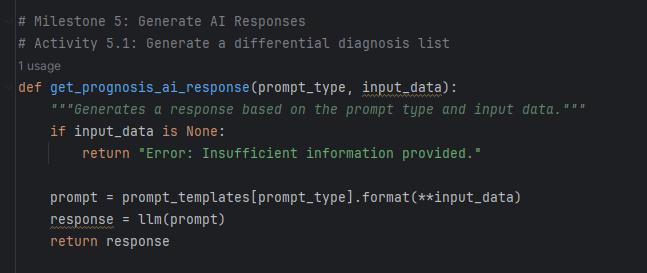
**Activity 5.1: Collect diagnostic support input from the user**

****

* This activity involves creating a form in Streamlit for users to input patient symptoms, medical history, and test results.
* Create a form using st.form() to collect patient symptoms, medical history, and test results.
* Use text areas for user input and a submit button to capture the information.
* Return the collected data in a dictionary format for further processing.
* The collected data is then used to generate a differential diagnosis.
* This activity gathers essential patient information via a user-friendly interface, which is then used to generate AI-driven diagnostic outputs.

**Milestone 6: Generate AI Responses**

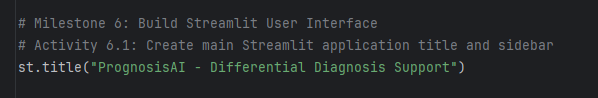
**Activity 6.1: Generate a differential diagnosis list**



* This activity involves using the AI model to generate a list of possible diagnoses, that match the patient's diagnosis, disease stage, and treatment history based on the input data.
* Format the input data into the prompt template.
* Call the Google GenAI model with the formatted prompt to generate a response.
* Handle errors and ensure the model returns meaningful differential diagnoses.
* The generated response includes potential risks, complications, and recommended additional tests for diagnosis.
* Using the AI model to analyze patient data and generate a list of potential diagnoses helps healthcare professionals quickly identify likely conditions and required further actions.

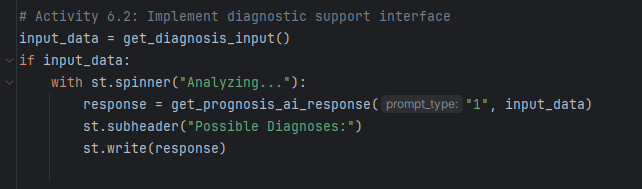
**Milestone 7: Build Streamlit User Interface**

**Activity 7.1: Create main Streamlit application title**



* This step involves setting up the main title of the Streamlit application.
* Input: Streamlit UI configuration.
* Process: Set up the main title and navigation options in the Streamlit app.
* Output: A navigable Streamlit application with a clear title and options.
* This step involves designing the main interface of the application, ensuring users can easily navigate to the diagnostic support functionalities.

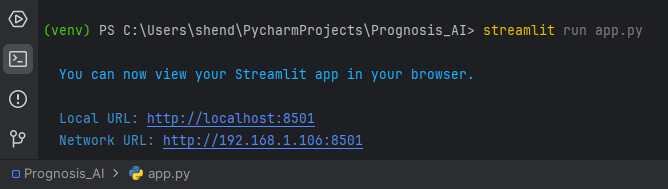
**Activity 7.2: Implement diagnostic support interface**



* Display a form for collecting diagnostic support input from the user.
* Show a loading spinner while the AI model processes the input.
* Display the AI-generated differential diagnosis list in a subheader.
* Input: User input form and AI model.
* Process: Display a form for input, show a spinner while processing, and then display the AI-generated diagnostic list.
* Output: An interactive interface that provides differential diagnoses based on user input.
* Implementing the diagnostic support interface allows users to interact with the AI, inputting patient data and receiving diagnostic suggestions in a streamlined manner.

**Activity 7.3: Run the application**

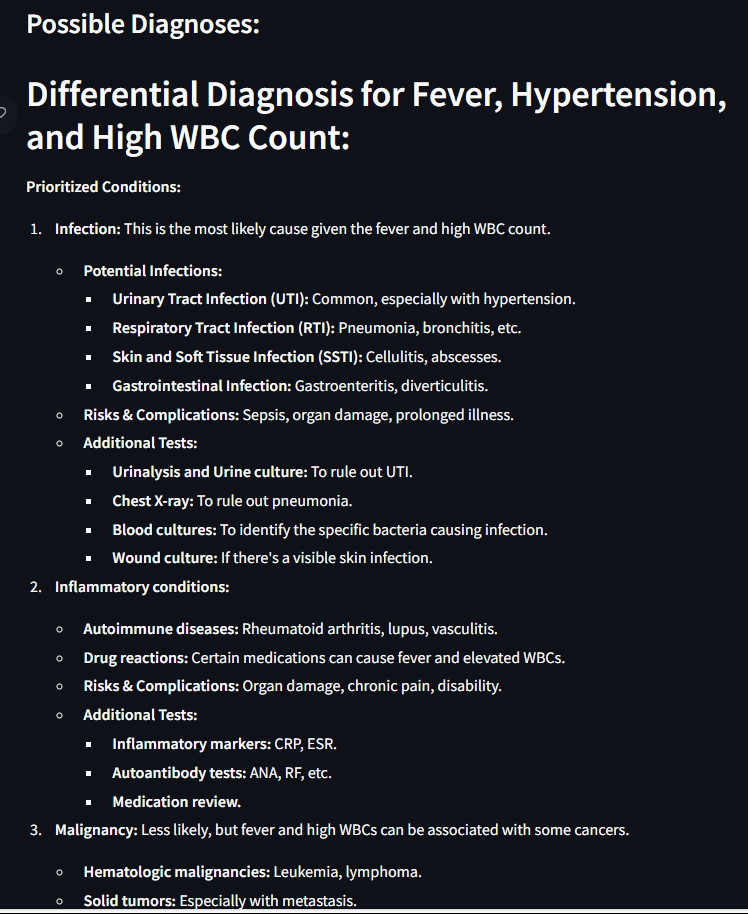
streamlit run app.py

****

* Input: Streamlit run command.
* Process: Execute streamlit run app.py in the terminal.
* Output: The application is deployed and accessible via a web browser.
* Running the application finalizes the development process, making the AI-driven diagnostic tool available for use by healthcare professionals.

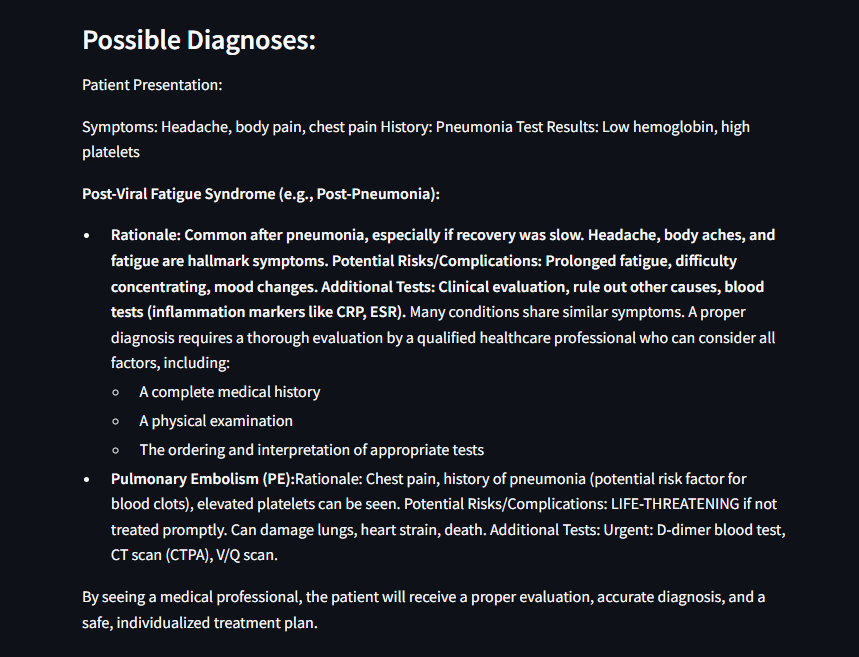
**Input 1**



****

**Input 2**

****

****