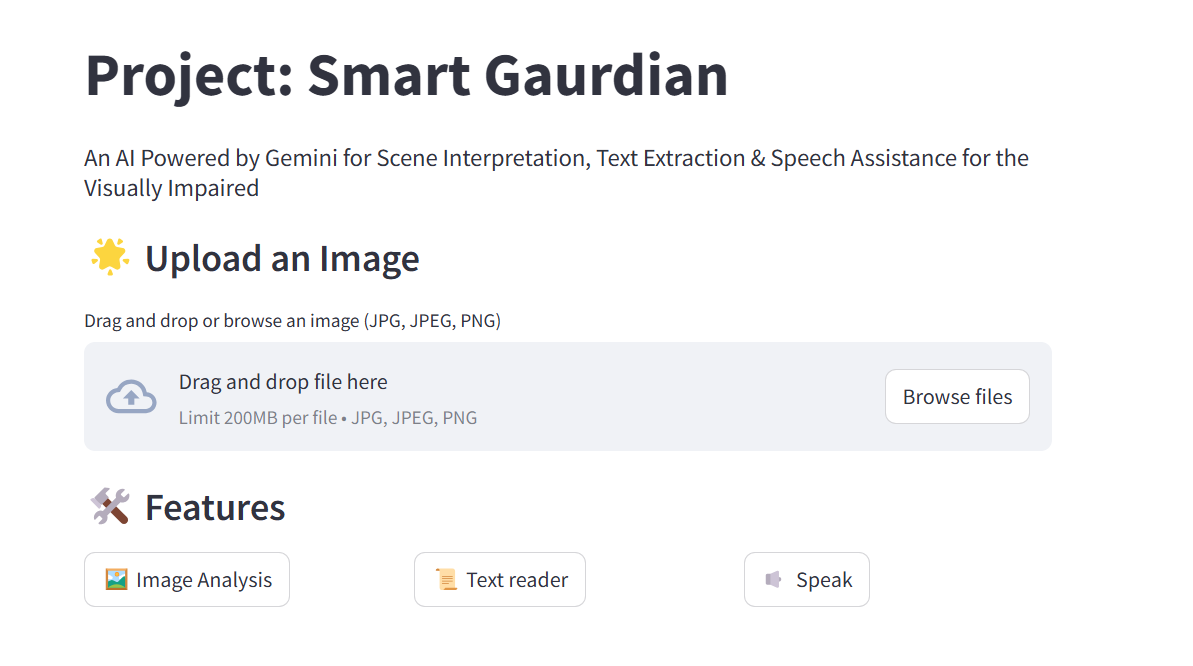
**Project: Smart Guardian Documentation**

**Introduction**

Smart Guardian is an AI-powered web application designed to assist visually impaired individuals. It leverages advanced technologies to analyze images, extract text, and provide voice-based assistance. This application is built using **Streamlit**, integrates Google Generative AI (Gemini model), and includes the LangChain library to enhance scene understanding.



**Features**

1. **Image Analysis**: Generates a detailed description of the uploaded image.
2. **Text reader**: Uses Optical Character Recognition (OCR) to extract text from images.
3. **Speak**: Converts text and scene descriptions into speech.

**Technologies Used**

1. **Streamlit**: For creating a user-friendly web interface.
2. **PyTesseract**: For OCR to extract text from images.
3. **Google Generative AI (Gemini)**: For generating detailed scene descriptions.
4. **LangChain**: For managing prompts and improving the AI's context understanding.
5. **pyttsx3**: For text-to-speech conversion.

**Code Explanation**

Below is a detailed breakdown of the code with explanations.

1. **Setting Up Dependencies**

import streamlit as st

import pyttsx3

import threading

from PIL import Image

import pytesseract

import os

import google.generativeai as genai

from langchain.prompts import PromptTemplate

* **Streamlit (st)**: Builds the app's interface.
* **pyttsx3**: A Python library for speech synthesis.
* **threading**: Ensures the text-to-speech process does not block the app.
* **Pillow (Image)**: Processes image files.
* **pytesseract**: Performs OCR for text extraction.
* **google.generativeai**: Connects to Google Gemini for content generation.
* **LangChain**: Creates structured prompts for AI tasks.

1. **Configuring External Tools**

**Tesseract OCR Path:**

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

* Specifies the location of the Tesseract OCR executable. Update this path based on your system setup.

**Google Generative AI API Key:**

genai.configure(api\_key="YOUR\_API\_KEY")

* **Replace "YOUR\_API\_KEY"** with your valid Google API key.

1. **Streamlit App Configuration**

st.set\_page\_config(page\_title=”Smart Gaurdian")

* Sets the title of the web app.

st.title("Project: Smart Guardian")

st.markdown(

"""

<h3 style='font-size:18px; font-weight:normal;'>

An AI Powered by Gemini for Scene Interpretation, Text Extraction & Speech Assistance for the Visually Impaired

</h3>

""",

unsafe\_allow\_html=True

)

* Displays the app title and a description.

1. **LangChain Prompt for Scene Analysis**

langchain\_prompt\_template = PromptTemplate(

input\_variables=["user\_context"],

template="""

As an AI assistant, you assist visually impaired users by interpreting the content of images.

User Context: {user\_context}

Please provide:

1. A list of identified items in the image along with their purposes/functions.

2. Overall description of the image.

3. Recommendations/Suggestions for actions or safety measures/precautions for the visually impaired.

"""

)

* **LangChain PromptTemplate**: Structures prompts for Google Generative AI to produce clear, user-focused scene descriptions.

**5. Functionality**

**Extract Text from Image**

def extract\_text\_from\_image(image):

"""Extracts text from the given image using OCR."""

return pytesseract.image\_to\_string(image)

* Uses PyTesseract to extract text from the uploaded image.

**Convert Text to Speech**

def text\_to\_speech(text):

"""Convert text to speech and speak it."""

def speak():

local\_engine = pyttsx3.init()

local\_engine.setProperty("rate", 150)

local\_engine.say(text)

local\_engine.runAndWait()

local\_engine.stop()

threading.Thread(target=speak, daemon=True).start()

* Converts text into spoken words using pyttsx3.
* **Threading** ensures this process doesn't block the app.

**Generate Scene Description**

def generate\_scene\_description(input\_prompt, image\_data):

"""Generates a scene description using Google Generative AI."""

model = genai.GenerativeModel("gemini-1.5-pro")

response = model.generate\_content([input\_prompt, image\_data[0]])

return response.text

* Sends a LangChain-generated prompt and image data to the Gemini model.
* Returns the AI-generated description.

**Prepare Uploaded Image**

def input\_image\_setup(uploaded\_file):

"""Prepares the uploaded image for processing."""

if uploaded\_file is not None:

bytes\_data = uploaded\_file.getvalue()

image\_parts = [

{

"mime\_type": uploaded\_file.type,

"data": bytes\_data,

}

]

return image\_parts

else:

raise FileNotFoundError("No file uploaded.")

* Processes the uploaded file into a format compatible with the AI model.

1. **User Interface**

**Image Upload Section**

uploaded\_file = st.file\_uploader("Drag and drop or browse an image (JPG, JPEG, PNG)", type=["jpg", "jpeg", "png"])

if uploaded\_file:

image = Image.open(uploaded\_file)

st.image(image, caption="Uploaded Image", use\_container\_width=True)

* Allows users to upload an image. Displays the uploaded image.

**Interactive Buttons**

scene\_button = col1.button("🖼️ Image Analysis")

ocr\_button = col2.button("📜 Text reader")

tts\_button = col3.button("🔈 Speak ")

* Provides buttons for:
  + **Image Analysis**: Generates a description.
  + **Text reader**: Extracts text using OCR.
  + **Speak**: Reads text aloud.

**7. Handling User Interactions**

**Scene Analysis**

if scene\_button:

with st.spinner("🖼️ Generating scene description..."):

scene\_description = generate\_scene\_description(input\_prompt, image\_data)

st.session\_state.scene\_description = scene\_description

st.markdown("### 🖼️ Scene Description")

st.success(scene\_description)

text\_to\_speech(scene\_description)

* Displays a loading spinner while generating a scene description.
* Saves the description to **session state** and reads it aloud.

**Text Extraction**

if ocr\_button:

with st.spinner("📜 Extracting text from the image..."):

ocr\_text = extract\_text\_from\_image(image)

st.session\_state.ocr\_text = ocr\_text

st.markdown("### 📜 Extracted Text")

st.text\_area("Extracted Text", ocr\_text, height=150)

text\_to\_speech(ocr\_text)

* Extracts text, saves it, and reads it aloud.

**Voice Narration**

if tts\_button:

if st.session\_state.scene\_description:

text\_to\_speech(st.session\_state.scene\_description)

elif st.session\_state.ocr\_text:

text\_to\_speech(st.session\_state.ocr\_text)

else:

st.warning("⚠️ Please generate a scene description or extract text first.")

* Speaks the available scene description or OCR text.

**How to Use**

1. **Upload an Image**: Upload a JPG, JPEG, or PNG file.
2. **Select Features**:
   * **Image Analysis**: Get a description of the image.
   * **Text reader**: Extract text from the image.
   * **Speak**: Hear the results read aloud.
3. **Listen to Outputs**: Use headphones for better clarity.

**Conclusion**

This project demonstrates how AI can assist visually impaired individuals by combining **image analysis**, **text extraction**, and **speech synthesis** into one seamless application.