AI-Driven Custom Home Design Assistant

The Custom Home Design Creator project focuses on developing an AI-powered tool for generating personalized home design plans. The objective is to build a generative model that creates custom home layouts based on user inputs such as preferences, spatial requirements, and architectural styles. By analyzing these inputs, the model produces detailed and aesthetically pleasing home designs that align with the user's vision and functional needs. This tool aims to simplify the home design process, allowing users to explore and visualize their ideal living spaces efficiently, while providing a creative and personalized approach to home planning and design.

Scenario 1: Real Estate Development

In a real estate development firm, the goal is to offer potential buyers customized home designs based on their preferences. Clients input their desired features, such as the number of bedrooms, architectural style, and special amenities. The Custom Home Design Creator generates personalized home layouts that match these specifications. This approach helps clients visualize their ideal homes and enhances their purchasing experience, similar to how tailored property showcases can drive interest and sales.

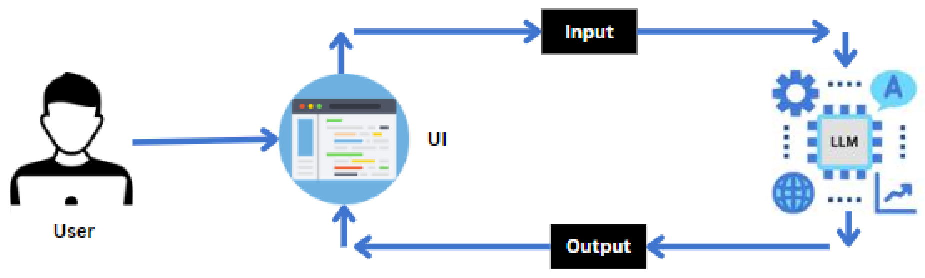
Scenario 2: Home Renovation Services

For a home renovation company, the tool is used to create design proposals for clients looking to remodel their existing homes. Users provide details about their current space, renovation goals, and style preferences. The Custom Home Design Creator then generates updated design plans that reflect these inputs, helping clients visualize potential changes. This facilitates the decision-making process and allows for more informed planning, akin to how virtual staging is used in real estate.

Scenario 3: Architectural Firm

In an architectural firm, the tool is utilized to quickly generate preliminary design concepts for clients based on their input. Architects and clients collaborate by specifying requirements such as room layout, design aesthetics, and functional needs. The Custom Home Design Creator produces detailed design drafts that can be refined further. This accelerates the design process and provides a clear starting point for discussions, similar to how initial sketches are used in architectural planning.

Architecture:



Project Flow

➢ **User Input via Streamlit UI:**

Users input a prompt (e.g., topic, keywords) and specify parameters such as the desired length, tone, or style through the Streamlit interface.

➢ **Backend Processing with Generative AI Model:**

The input data is sent to the backend, where it interfaces with the selected Generative AI model (e.g., GPT-4, Gemini, etc.).

The model processes the input, generating text based on the specified parameters and user input.

➢ **Content Generation:**

The AI model autonomously creates content tailored to the user's specifications. This could be a blog post, poem, article, or any other form of text.

➢ **Return and Display Generated Content:**

The generated content is sent back to the frontend for display on the Streamlit app.

The app presents the content to the user in an easily readable format.

➢ **Customization and Finalization:**

Users can further customize the generated content through the Streamlit UI if desired. This might include editing text, adjusting length, or altering tone.

➢ **Export and Usage:**

Once satisfied, users can export or copy the content for their use, such as saving it to a file or directly sharing it.

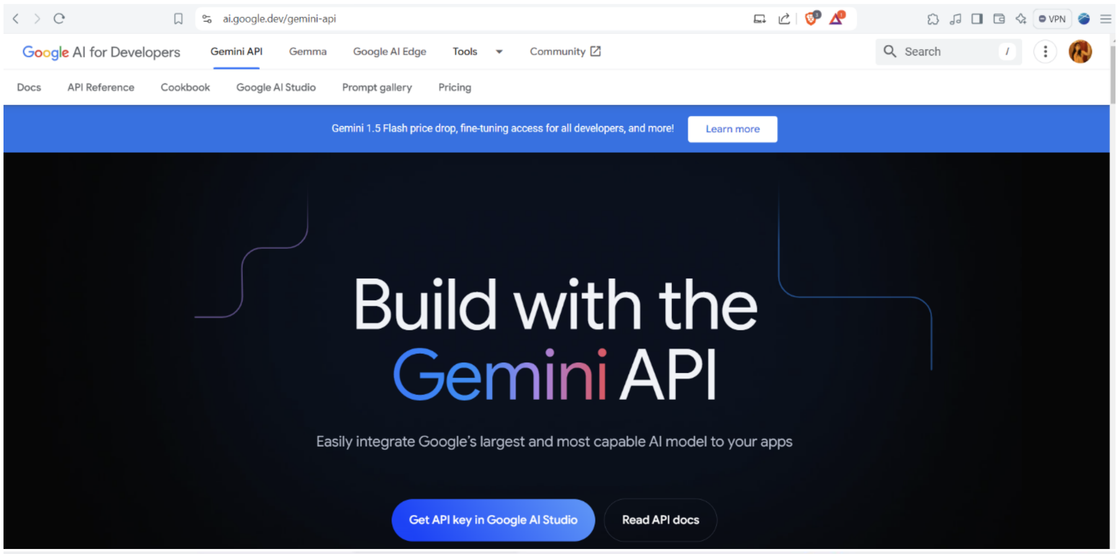
Requirements Specification

Install the libraries

- pip install streamlit
- pip install google.generativeai

Initializing the Models

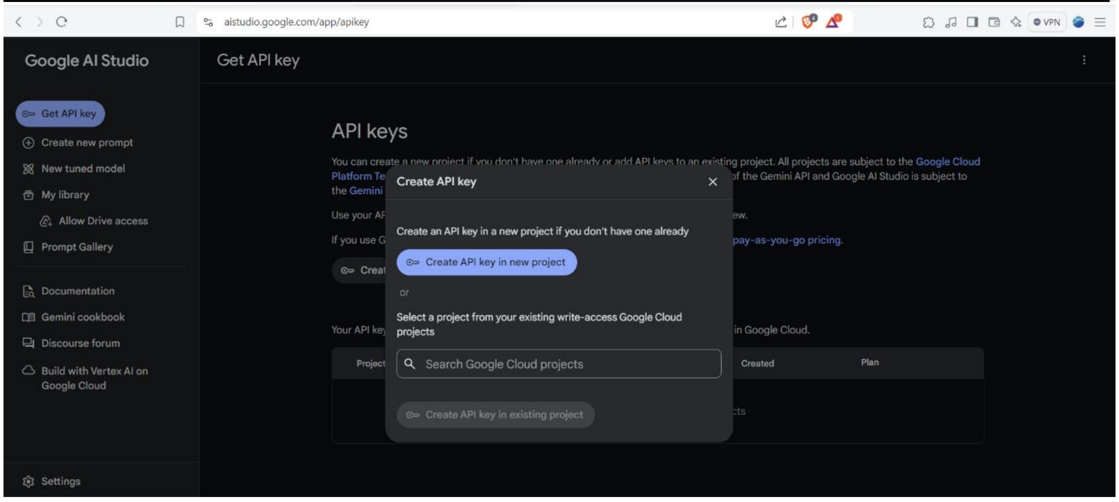
Generating API



Gemini 1.5 Flash price drop, fine-tuning access for all developers, and more!

Build with the Gemini API

Easily integrate Google's largest and most capable AI model to your apps



Create an API key in a new project if you don't have one already

Select a project from your existing write-access Google Cloud projects

Link: https://ai.google.dev/gemini-api

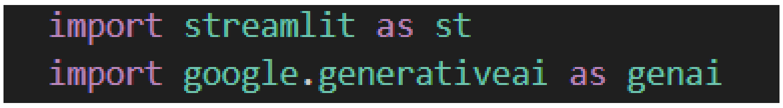
Enable the Gemini API

- Once your project is created, navigate to the API & Services Dashboard.
- Click on Enable APIs and Services at the top.
- Search for "Gemini API" in the API library.
- Select the Gemini API and click Enable.

Initialize the pre-trained model

- Streamlit, a popular Python library, is imported as st, enabling the creation of user interfaces directly within the Python script.
- Google Generative AI (genai): Imported to interact with the Gemini Pro model.

Activity 2.1: Importing Libraries

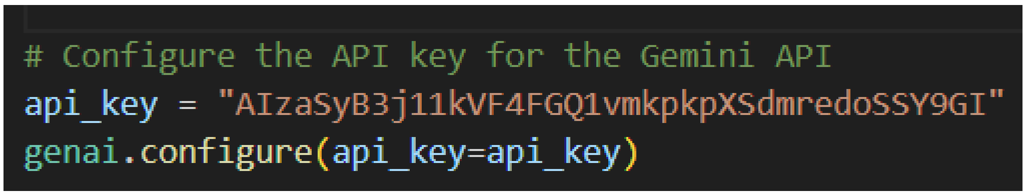


```
import streamlit as st
import google.generativeai as genai
```

1. Streamlit (streamlit as st): This is a popular open-source framework used for creating web apps with Python. The st alias allows you to easily access Streamlit's functions for building user interfaces, like adding buttons, displaying data, and more.

2. Google's Generative AI (google.generativeai as genai): This module likely provides access to Google's generative AI models. The genai alias allows you to interact with these models to generate text, images, or other outputs, depending on what Google's generative AI API offers.

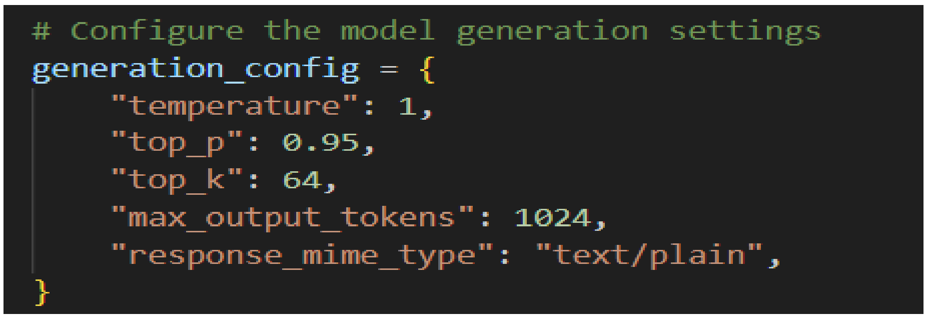
Activity 2.2: Configurating of the Gemini Pro API



```
# Configure the API key for the Gemini API
api_key = "AIzaSyB3j11kVF4FGQ1vmkpkpXSdmredoSSY9GI"
genai.configure(api_key=api_key)
```

This code sets an API key for accessing Google's Generative AI services. The api_key variable stores the key, and genai.configure(api_key=api_key) configures the genai module to use this key for authenticating requests to Google's Generative AI API. This allows secure interaction with the AI models.

Activity 2.3: Defining the model



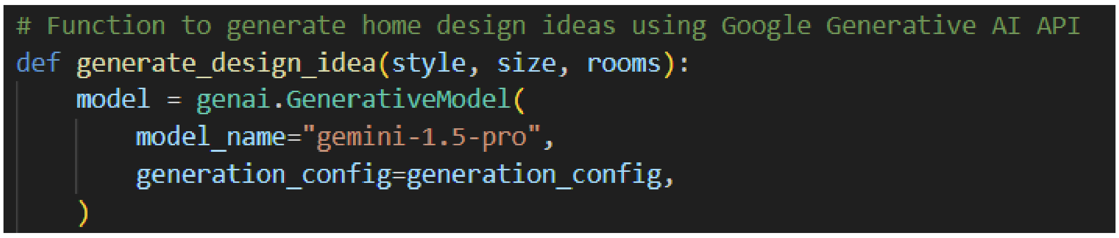
```
# Configure the model generation settings
generation_config = {
    "temperature": 1,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 1024,
    "response_mime_type": "text/plain",
}
```

This snippet defines a dictionary called generation_config that sets various parameters for generating outputs using a generative AI model:

- ➢ temperature: Controls randomness. A value of 1 provides balanced randomness in the generated text.
- ➢ top_p: A sampling method that selects tokens from the smallest set whose probabilities sum up to 0.95, ensuring diversity.
- ➢ top_k: Limits the selection to the top 64 tokens, further controlling diversity.
- ➢ max_output_tokens: Specifies the maximum length of the generated text, capped at 1024 tokens.
- ➢ response_mime_type: Indicates the format of the response, here set to plain text.

Interfacing with Pre-trained Model

Creating Function & Defining the model



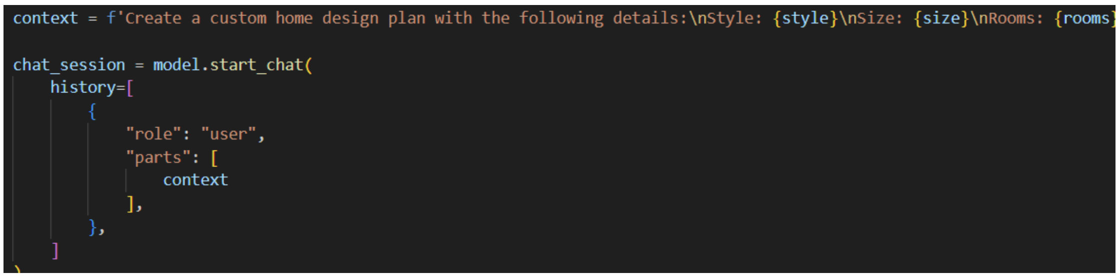
```python
# Function to generate home design ideas using Google Generative AI API
def generate_design_idea(style, size, rooms):
    model = genai.GenerativeModel(
        model_name="gemini-1.5-pro",
        generation_config=generation_config,
    )
```

This function, generate_design_idea, is designed to create a custom home design plan based on user inputs. It takes three parameters:

- style: The style of the home design (e.g., Modern, Rustic).
- size: The size of the home in square feet or another unit.
- rooms: The number of rooms in the home.

model_name: Specifies which AI model to use (in this case, "gemini-1.5-pro").

generation_config: A dictionary of settings that guide how the model generates the text (e.g., temperature, top_p, etc.).



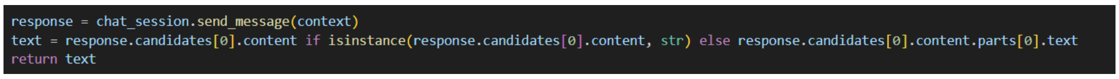
```python
context = f'Create a custom home design plan with the following details:\nStyle: {style}\nSize: {size}\nRooms: {rooms

chat_session = model.start_chat(
    history=[
        {
            "role": "user",
            "parts": [
                context
            ],
        },
    ]
```

Context Definition:

- A context string is defined to specify the requirements for generating a custom home design plan. It includes details such as style, size, and number of rooms. It also outlines additional aspects to include, like layout suggestions, color schemes, and furniture recommendations, and requests the response in Markdown format.

Chat Session Initialization:

- A chat session is started with a model (e.g., an AI model) using the start_chat method.
- The history parameter is used to provide initial input to the model. In this case, it consists of a single message from the user that includes the context for the conversation.
- This setup provides the model with the necessary information to generate a response based on the specified requirements.



```python
response = chat_session.send_message(context)
text = response.candidates[0].content if isinstance(response.candidates[0].content, str) else response.candidates[0].content.parts[0].text
return text
```

Send Message:

- A message containing the context is sent to the chat session, which requests the model to generate a response based on the provided information.

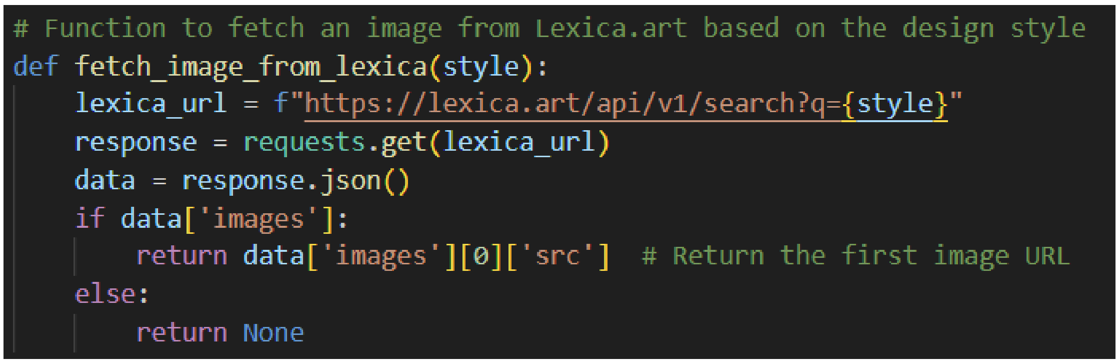
Process Response:

- The code checks the format of the response content. If it's a straightforward string, it assigns it directly to a variable.
- If the response content is more complex (e.g., a structured object), it extracts the relevant text from the first part of the content.

Return Text:

- The extracted or processed text is returned, which represents the model's generated response.

Activity 2: Creating Function for image



```python
# Function to fetch an image from Lexica.art based on the design style
def fetch_image_from_lexica(style):
    lexica_url = f"https://lexica.art/api/v1/search?q={style}"
    response = requests.get(lexica_url)
    data = response.json()
    if data['images']:
        return data['images'][0]['src']  # Return the first image URL
    else:
        return None
```

Function Definition:

- The function fetch_image_from_lexica is designed to retrieve an image from Lexica.art based on a specified design style.

Construct URL:

- lexica_url is created by embedding the style parameter into the URL. This URL is used to query the Lexica.art API for images that match the given design style.

Send Request:

- A GET request is sent to the Lexica.art API using the constructed URL. This request fetches data from the API.

Parse Response:

- The response from the API is converted from JSON format to a Python dictionary.

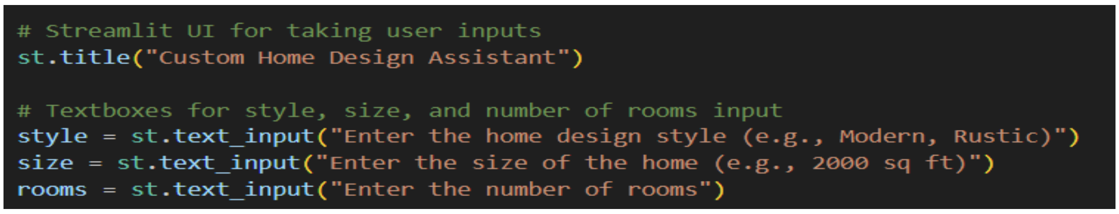
Check and Return Image URL:

- The function checks if there are any images in the response. If images are present, it returns the URL of the first image.
- If no images are found, it returns None.

Model Deployment

- In this milestone, we deploy the created model using Streamlit. Streamlit allows us to create a user-friendly web interface, enabling users to interact with the model through their web browser

Starting Streamlit



```python
# Streamlit UI for taking user inputs
st.title("Custom Home Design Assistant")

# Textboxes for style, size, and number of rooms input
style = st.text_input("Enter the home design style (e.g., Modern, Rustic)")
size = st.text_input("Enter the size of the home (e.g., 2000 sq ft)")
rooms = st.text_input("Enter the number of rooms")
```

Title:

- Displays the title of the Streamlit app, "Custom Home Design Assistant."

Text Input for Style:

- Provides a text input field for users to enter the design style of the home.

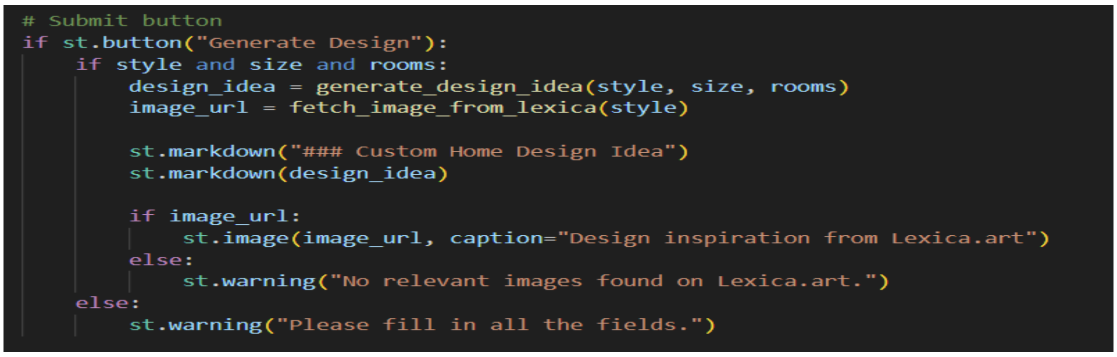
Text Input for Size:

- Provides a text input field for users to enter the size of the home.

Text Input for Number of Rooms:

- Provides a text input field for users to enter the number of rooms in the home.

Displaying for user



```python
# Submit button
if st.button("Generate Design"):
    if style and size and rooms:
        design_idea = generate_design_idea(style, size, rooms)
        image_url = fetch_image_from_lexica(style)

        st.markdown("### Custom Home Design Idea")
        st.markdown(design_idea)

        if image_url:
            st.image(image_url, caption="Design inspiration from Lexica.art")
        else:
            st.warning("No relevant images found on Lexica.art.")
    else:
        st.warning("Please fill in all the fields.")
```

Submit Button:

- A button labeled "Generate Design" is displayed to the user.

Button Action:

- When the button is clicked, the code checks if all input fields (style, size, and rooms) are filled.

Generate Design Idea:

- If all fields are filled, it calls a function to generate a design idea based on the provided inputs.

Fetch Image:

- It then fetches an image related to the design style from Lexica.art.

Display Design Idea:

- The generated design idea is displayed in the app using Markdown.

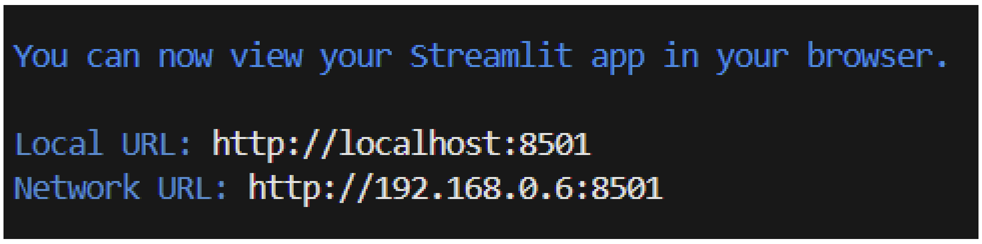
Display Image:

- If an image URL is returned, the image is displayed with a caption. If no image is found, a warning message is shown instead.

Field Validation:

- If any of the input fields are empty, a warning message prompts the user to fill in all fields.

Running the web application



```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.0.6:8501
```

- The application is now running and can be accessed locally through the provided URL. It is also available on the network via the network URL, allowing access from other devices on the same network.

Conclusion

The Travel Itinerary Planner project showcases the transformative impact of AI on travel planning. By harnessing Google Generative AI and integrating it with Streamlit, the tool offers a sophisticated and user-friendly solution for crafting personalized travel itineraries. The ability to input destination, trip duration, and activity preferences results in tailored travel plans that include detailed daily schedules, sightseeing opportunities, and dining options. This project not only streamlines the travel planning process but also enhances the overall travel experience by delivering well-organized and customized recommendations. The successful implementation of AI in this context underscores its potential to revolutionize personal planning across various domains.