



Project Name:- AI-Driven Custom Home Design Assistant

Team ID: LTVIP2025TMID27302

Team Size: 4

Team Leader: B Uma Maheshwar

Team Member: Katipogula Urmila

Team Member: Mankar Srinivas Sai

Team Member: Katta Pavithra

College Name: The Adoni Arts and Science College , Adoni

University: Rayalaseema University

AI-Driven Custom Home Design Assistant

Project Description

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.

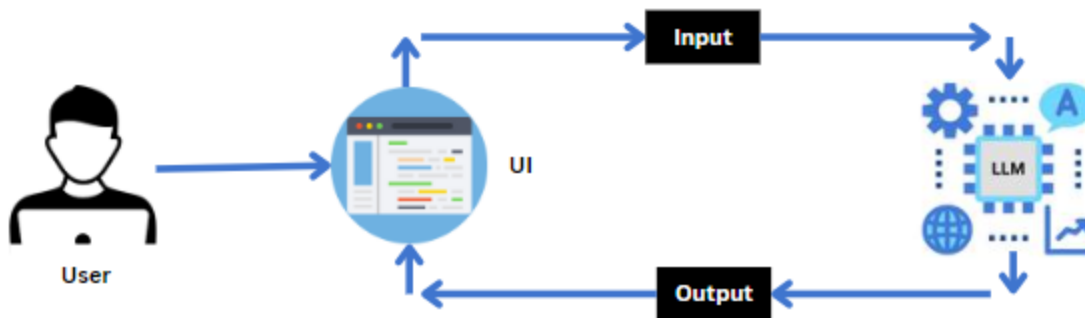
Scenario 2: Home Renovation Services

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.

Prior Knowledge

To work on a poetry generator project using Streamlit and Google Generative AI, prior knowledge in the following areas is essential:

1. Natural Language Processing (NLP): Understanding the basics of NLP is crucial for processing and generating text. Familiarity with concepts such as tokenization, text generation, and language models is necessary to build a foundation for the project.
2. Generative AI Models: Knowledge of generative models, particularly those like Google's language models, is important. You should understand how these models work, how to fine-tune them, and how to manage parameters like temperature and max tokens to control the output.
3. Python Programming: Proficiency in Python is required as it is the primary language used in both Streamlit and Google Generative AI. You should be comfortable writing scripts, handling APIs, and processing data.
4. Streamlit Framework: Familiarity with Streamlit is necessary to build the user interface for the poetry generator. You should know how to create interactive web applications, manage user inputs, and display generated text dynamically.
5. APIs and Integration: Understanding how to integrate different tools and services using APIs is important. This includes managing API requests, handling responses, and ensuring the seamless operation of the app.

Requirements Specification

Install the libraries

```
pip install streamlit
```

```
pip install google.generativeai
```

Initializing the Models

Generating API

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
import requests
```

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: Configuring of the API keys

```
# Configure the API key for the Gemini API
api_key = "AIzaSyCteWjKt80SY1yL0aZv7zQTMWJhE6y5JHI"
genai.configure(api_key=api_key)

# Pexels API Key (Replace with your own key from https://www.pexels.com/api/)
PEXELS_API_KEY = "P8w5PqSGzmM8dHr6D2ns0ldFBfUfp08pcoLDHyR1lPecey7IqeyxtqNq"
```

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. and Pexels API key is used for fetching images

Activity 2.3: Defining the model

```
# Configure the model generative settings
generative_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

```
# 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=generative_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.).

```
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.

```
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


```

Function to fetch an image from Pexels based on the design style
def fetch_image_from_pexels(query):
    url = f"https://api.pexels.com/v1/search?query={query}&per_page=1"
    headers = {"Authorization": PEXELS_API_KEY}
    try:
        response = requests.get(url, headers=headers)
        data = response.json()

        if "photos" in data and len(data["photos"]) > 0:
            return data["photos"][0]["src"]["large"] # Return first image URL
        else:
            return None

```

Function Definition:

The function `fetch_image_from_Pexels` is designed to retrieve an image from Pexels(query) based on a specified design style.

Construct URL:

`Pexels_url` is created by embedding the style parameter into the URL. This URL is used to query the Pexels API for images that match the given design style.

Send Request:

- A GET request is sent to the Pexels 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

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

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

Activity 2: Displaying for user

```

Submit button
st.button("🔍 Generate Design"):
    if style and size and rooms:
        design_idea = generate_design_idea(style, size, rooms)
        image_url = fetch_image_from_pexels(f"{style} home design")

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

        if image_url:
            st.image(image_url, caption="Design inspiration from Pexels")
        else:
            st.warning("No relevant image found on Pexels.")
    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.

Displaying for user

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.

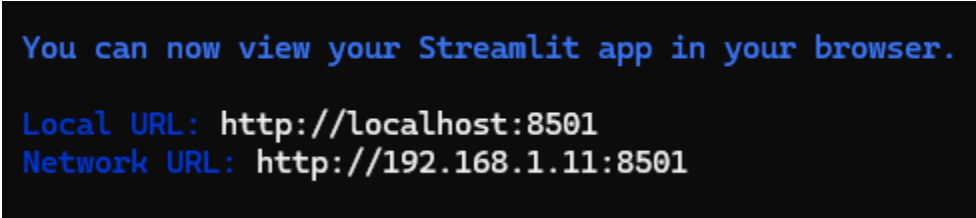
Display Image:

- 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

A terminal window with a black background and blue and white text. The text indicates that the Streamlit app is running and provides local and network URLs for access.

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

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


```
Network URL: http://192.168.1.11:8501
```

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.


Output




Custom Home Design Assistant

 Enter the home design style (e.g., Modern, Rustic)


Modern


 Enter the size of the home (e.g., 2000 sq ft)

3000

 Enter the number of rooms

2

 **Generate Design**



Custom Home Design Idea

parts { text: "## Modern 3000 sq ft, 2-Bedroom Home Design\n\nThis design prioritizes open-concept living, maximizing space and natural light with a focus on minimalist aesthetics and functionality for a modern lifestyle.\n\n**Concept:** A U-shaped design around a central courtyard provides privacy and brings the outdoors in. The two bedrooms are positioned for maximum seclusion, while the main living spaces embrace the courtyard view.\n\n**Exterior:**\n\n* **Materials:** Concrete, glass, and natural wood accents. Flat roof with integrated rainwater collection system.\n\n* **Landscaping:** Drought-tolerant landscaping with a focus on texture and native plants. The courtyard features a small reflecting pool and minimalist seating area.\n\n* **Parking:** Two-car garage with electric vehicle charging station. Additional driveway parking for guests.\n\n**Interior:**\n\n* **Entryway:** Double-height ceiling with floor-to-ceiling windows overlooking the courtyard. Built-in storage and bench seating.\n\n* **Great Room:** Open-concept living, dining, and kitchen area. Floor-to-ceiling glass doors open onto the courtyard. Fireplace feature wall.\n\n* **Kitchen:** Minimalist design with integrated appliances. Large island with waterfall countertop and seating. Walk-in pantry.\n\n* **Dining Area:** Located adjacent to the kitchen, with space for a large dining table. Direct access to the courtyard.\n\n* **Master Suite:** Located on one wing of the U-shape for maximum privacy. Features a spacious bedroom with courtyard views, a walk-in closet with built-in organizers, and a luxurious ensuite bathroom with a soaking tub, separate shower, and double vanity. Private access to a small outdoor patio.\n\n* **Guest Suite/Multipurpose Room:** Located on the opposite wing, this flexible space can be used as a guest bedroom, home office, or studio. Features a built-in Murphy bed for guest accommodations and



The design for a Modern/Rustic-style home of 3000 sq ft features a warm and inviting atmosphere using natural materials such as wood and stone. The layout emphasizes spaciousness with a cozy, traditional charm. Earthy color schemes with warm tones are used throughout. Key elements include a large central fireplace, exposed wooden beams, and rustic furniture. Large windows are incorporated to maximize natural light and connect the indoor and outdoor spaces. This design blends comfort and style to create a welcoming and timeless Modern/Rustic home.

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.