## **Building a Gemini AI-based Streamlit Chat Application**

### **Before you begin you will need the following:**

To get started, you’ll need to have a standard Google account. If you haven’t set one up yet, please visit Google’s account creation page to register for a new account. This will be essential for accessing your project's full range of features and services. Make sure you have created an account at <https://accounts.google.com/SignUp>.

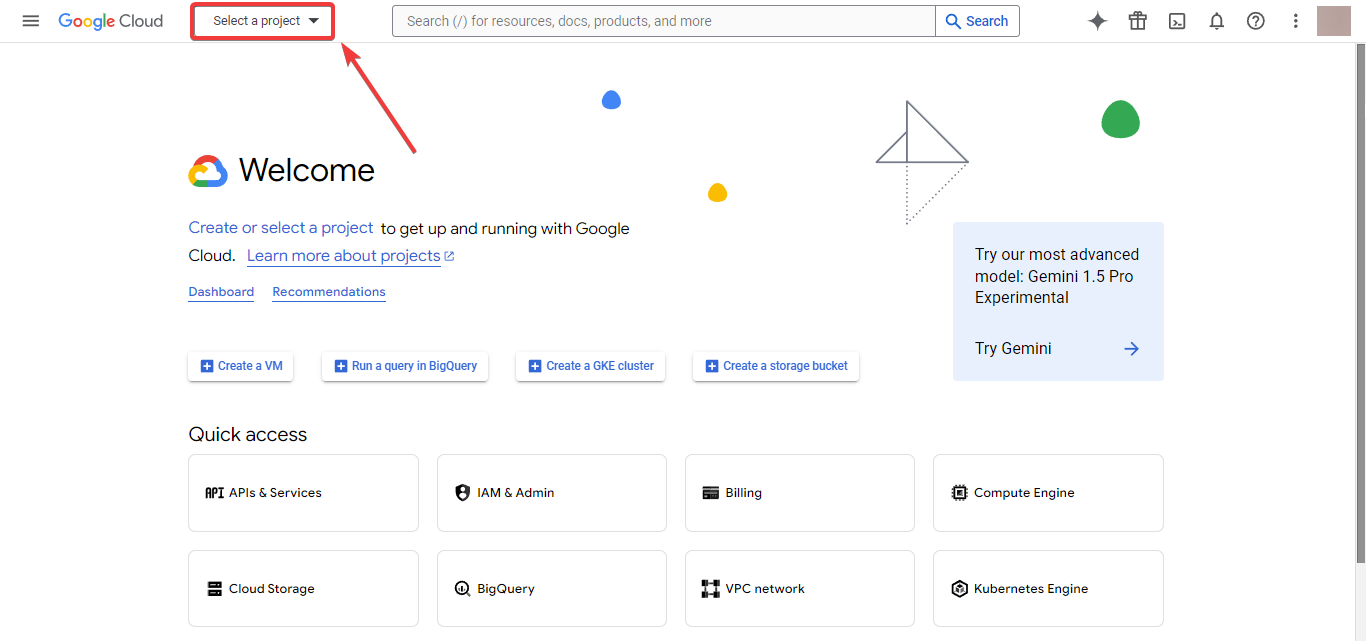
## **Creating a Google Developer Account (Play Console Account):**

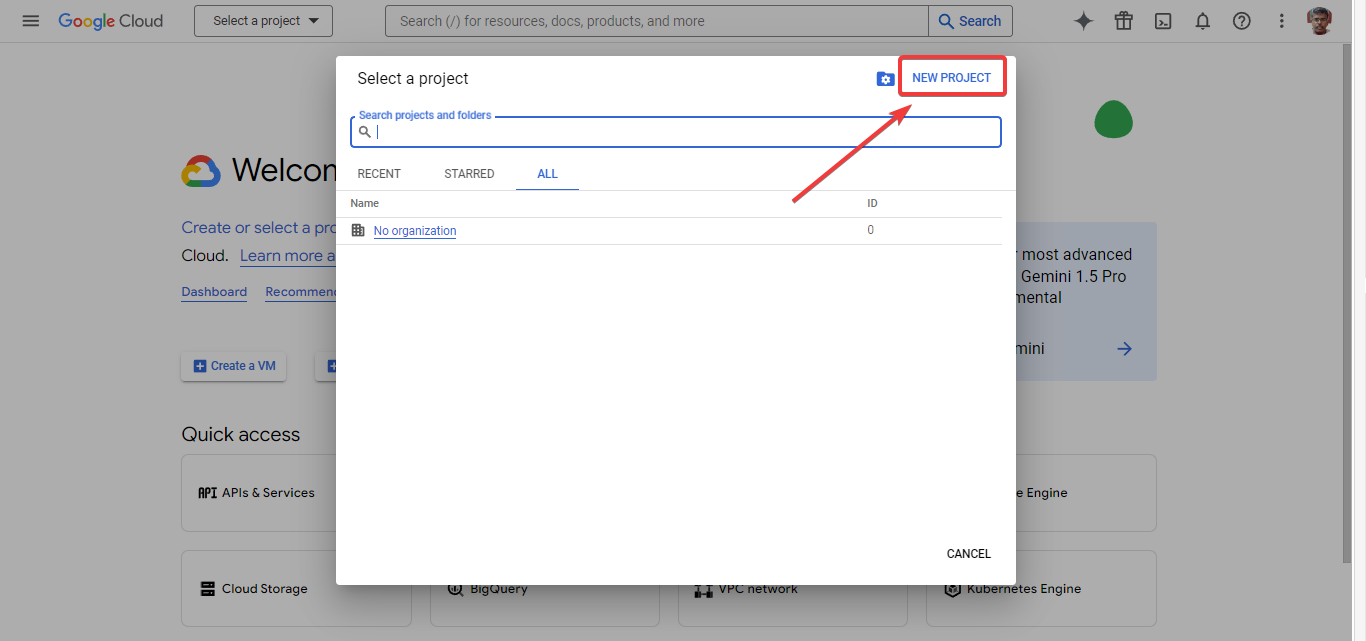
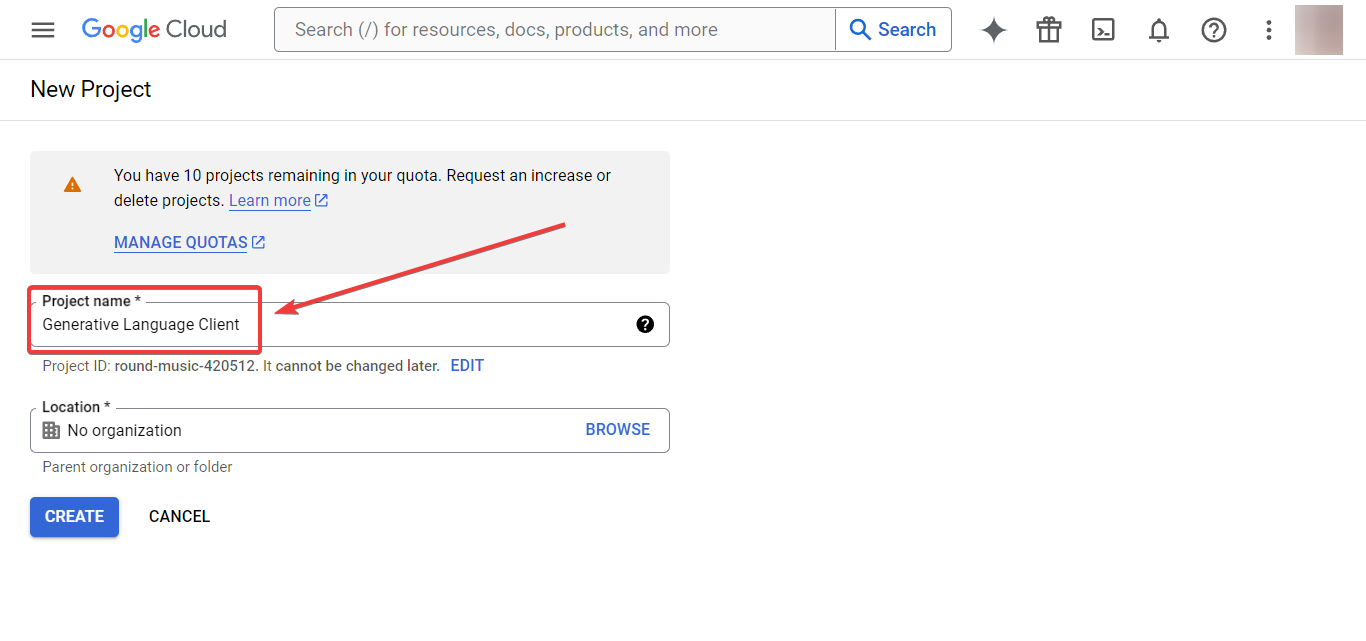
Please note that while creating a Google Developer account (Play Console account) typically requires a $25 registration fee, you can directly create a project in the Google Cloud platform console and obtain an API key from the Google AI Studio for our project without this fee.

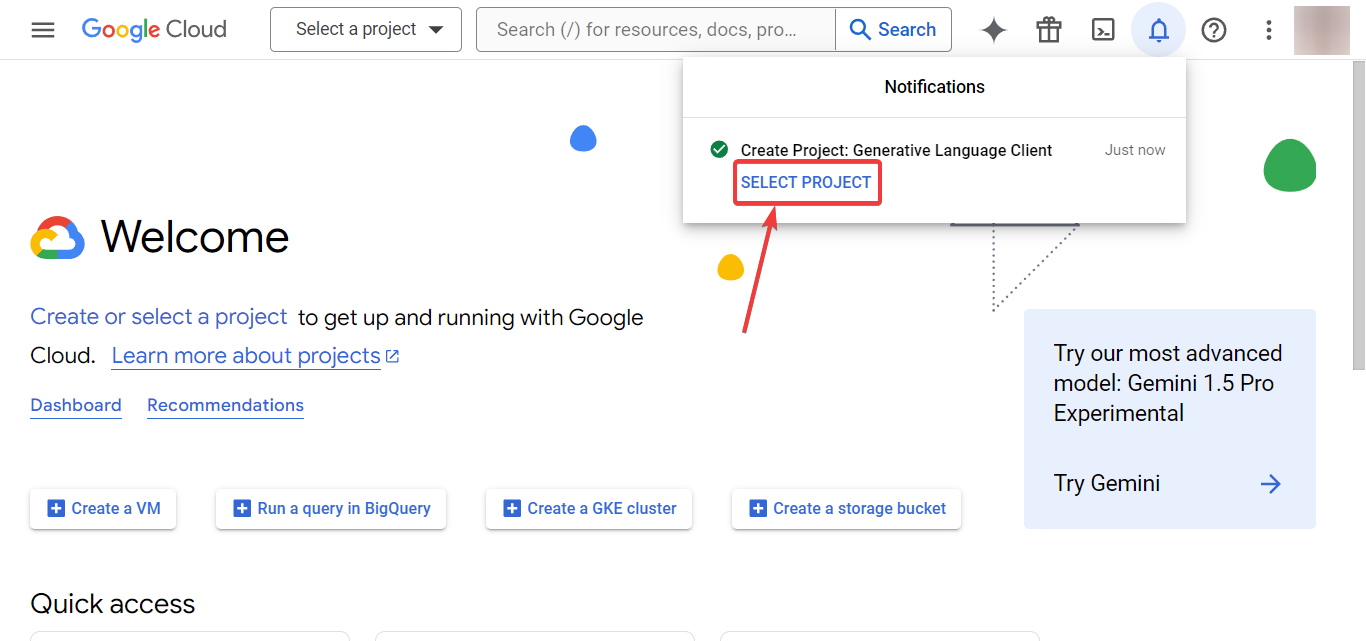
## **Creating a Project in the Google Cloud Platform Console:**

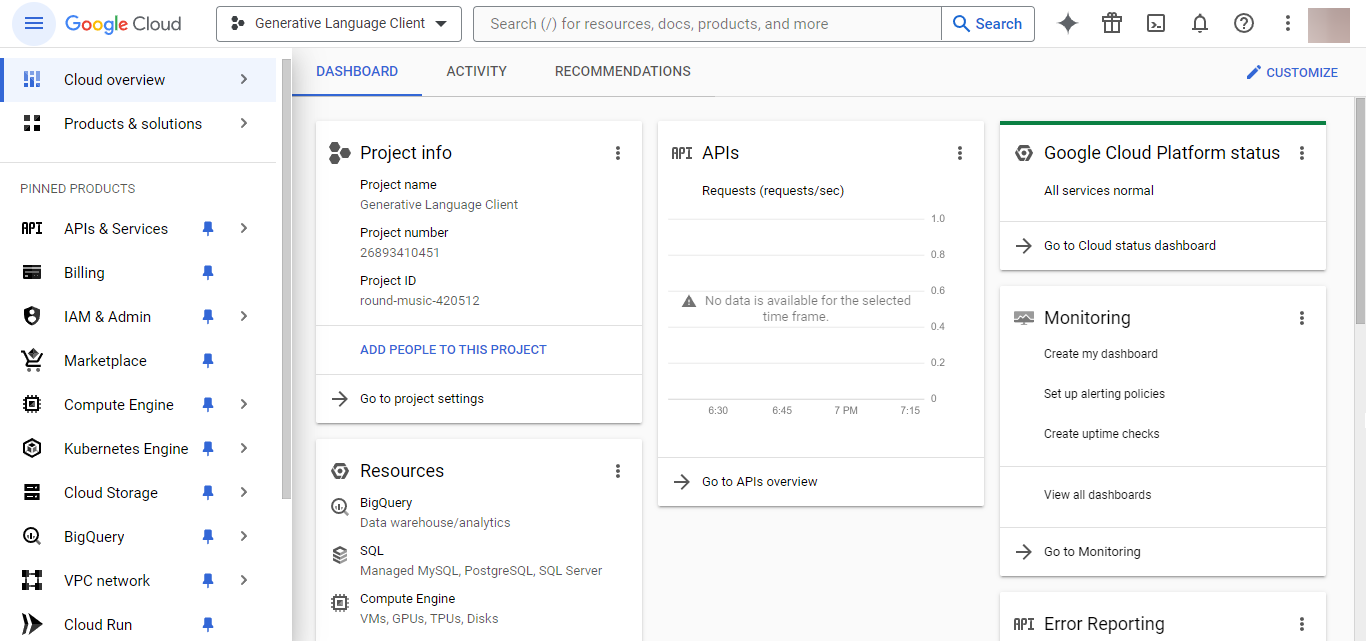
Creating a New Project in the Google Cloud Console:

1. Go to the Google Cloud Console platform, [https://console.cloud.google.com/](https://console.cloud.google.com/home/dashboard)
2. In the **Google Cloud Platform interface**, click on the 'Select a Project' button located at the top-left corner, adjacent to the Google Cloud icon.



1. In the **Select a Project** Dialog Box, Click on the **New Project** Button:
2. In the **New Project** creation page, Enter the Project Name that you want (*For Example Generative Language Client*) and Click on **Create**
3. You will be redirected to the welcome page of the console, where on the notifications area, you see the notification of our project created, Click **SELECT PROJECT.**

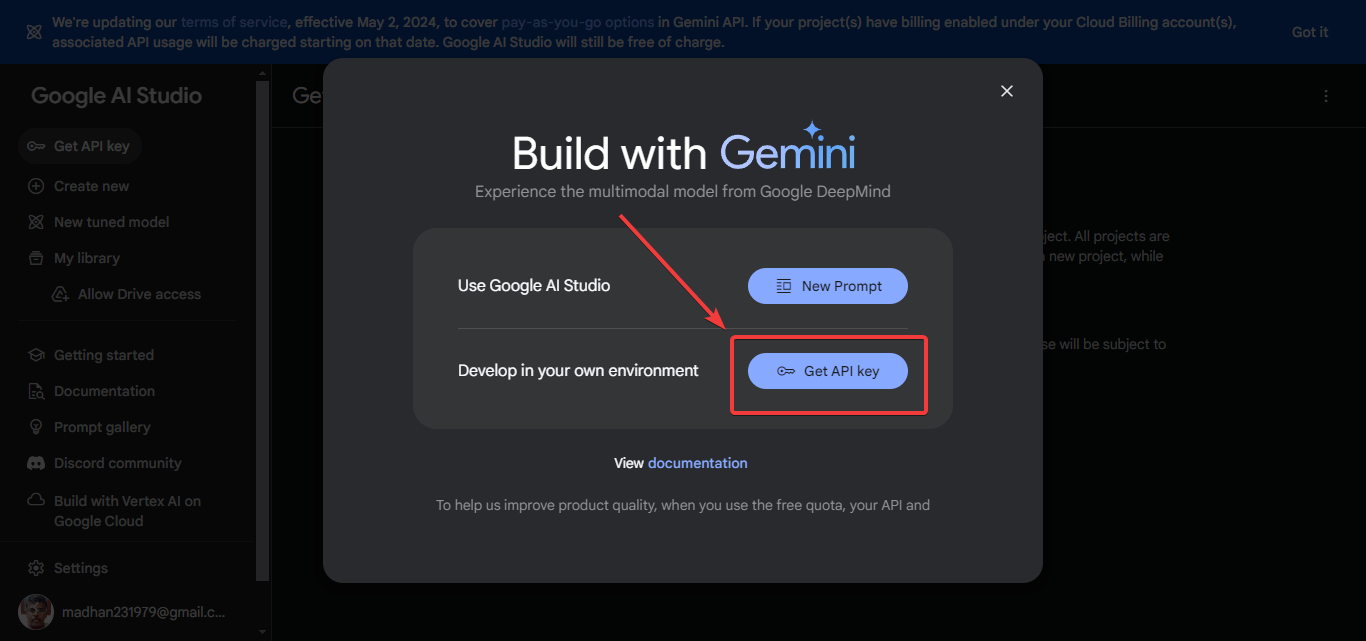
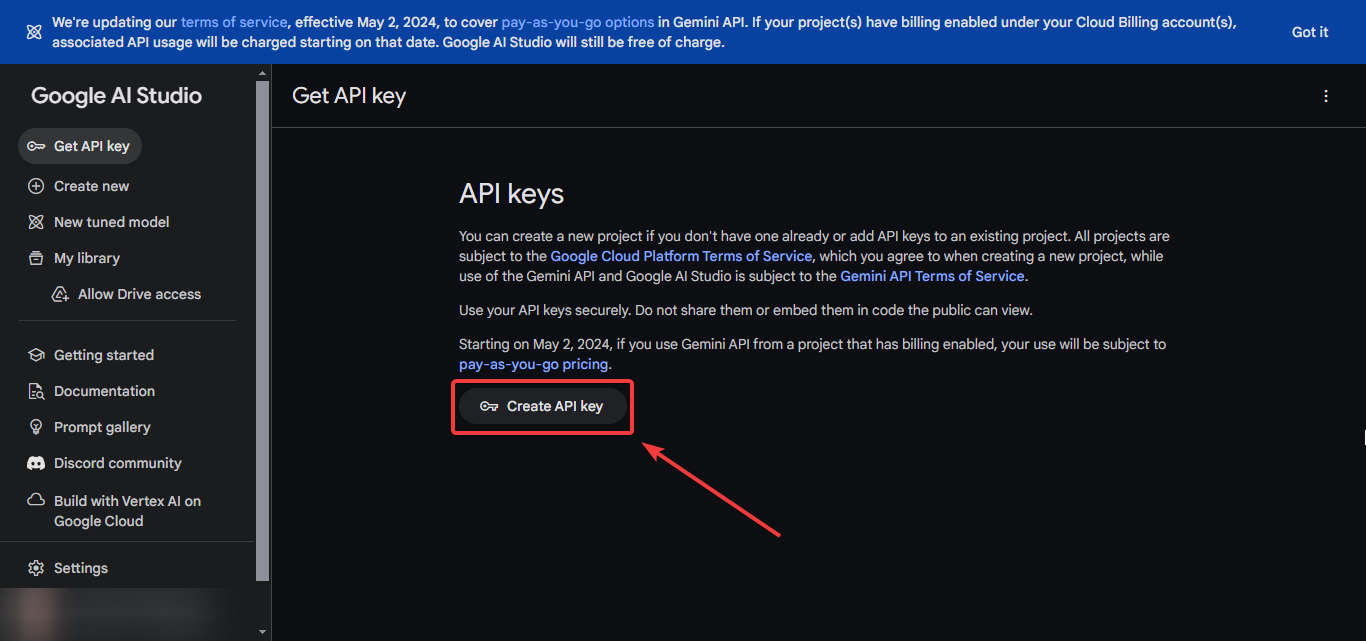
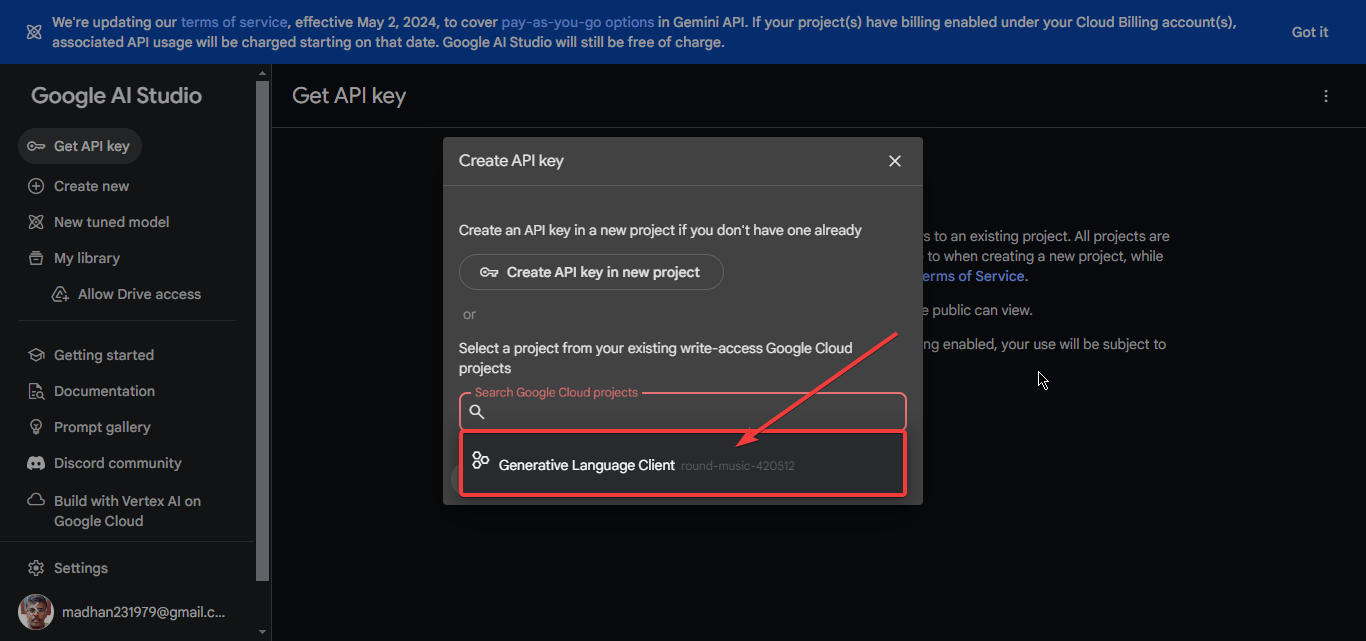
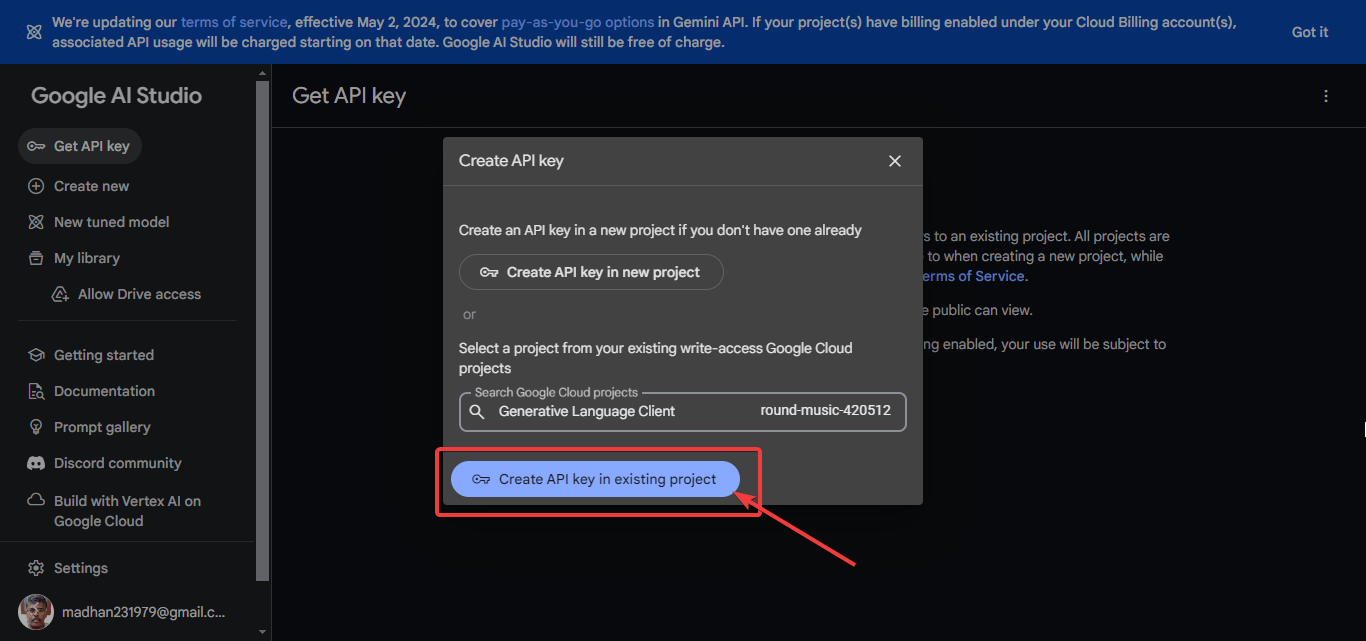
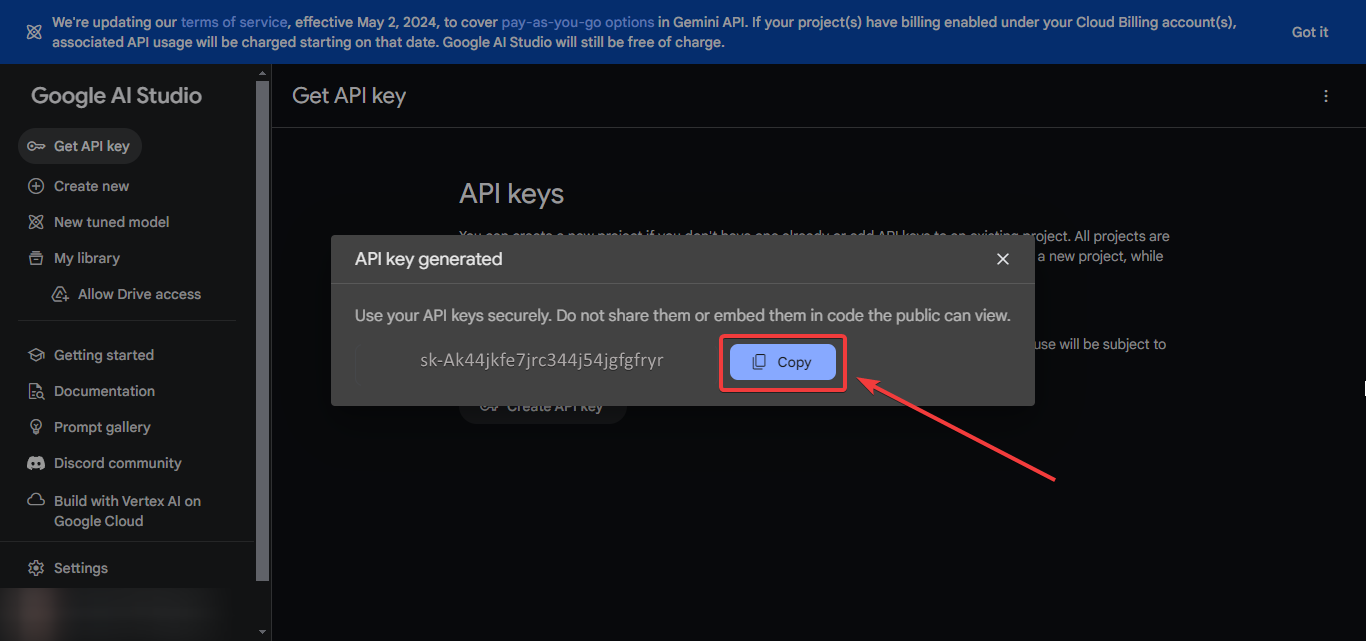


6. After Clicking the **SELECT PROJECT** you will be redirected to the dashboard of your project in the console

### **Creating an API Key from Google AI Studio:**

1. Go to **Google AI Studio**’s API key page (sign in with your Google account that you use to create the project in the GCP console):

<https://aistudio.google.com/app/apikey>

1. Where on the welcome dialogue box, click on the **Get API key** button  
   
2. On the API Key page click on the **Create API Key** button:
3. In the Create API Key dialogue box, select the project that we created on the GCP:
4. Then Click on **Create API Key in the existing project** button:
5. Then the API key for your project gets created, where click the **Copy** button:

**After Creating the API Key, secure it as recommended. Remember to use your API keys securely and follow best practices to protect them.**

## **Building a Streamlit Application with Gemini-Pro 1.0 Model:**

### **Installation Guide for Gemini API Python SDK:**

To begin using the Gemini API’s Python SDK, you’ll need to install the [google-generativeai](https://pypi.org/project/google-generativeai/) package. This package contains all the necessary tools to integrate Gemini API into your Python applications. Follow these steps to install the package:

1. **Open Your Command Line Interface**: Access your terminal on macOS/Linux or Command Prompt/PowerShell on Windows.

2. **Install the Package**: Execute the following command:

| pip install -q -U google-generativeai |
| --- |

This command will download and install the [google-generativeai](https://pypi.org/project/google-generativeai/) package from the Python Package Index (PyPI). Once the installation is complete, you can import the package into your Python scripts and start building the Gemini AI based Streamlit Chat Application 😊.

**Let’s Start Building the Streamlit Chat Application with Gemini:**

**Importing the necessary libraries:**

| import streamlit as st import google.generativeai as genai |
| --- |

**This step involves adding the required Python libraries to your script. It’s the foundation of your project, where you include packages that provide functionalities essential for your application, such as Streamlit for web app development and the Gemini API for AI-powered features.**

**Configuring the Gemini API:**

| # Configure the Gemini API GOOGLE\_API\_KEY = 'ENTER\_YOUR\_API\_KEY\_HERE' genai.configure(api\_key=GOOGLE\_API\_KEY) model = genai.GenerativeModel('gemini-pro') |
| --- |

**Here, you’ll set up the Gemini API by initializing it with your unique API key. This configuration is crucial as it authenticates your application’s requests to the Gemini services, allowing you to access its AI capabilities.**

**Streamlit page configuration and title setup:**

| # Set Streamlit page configuration st.set\_page\_config(  page\_title="Gemini Clone",  page\_icon="✨",  layout="wide" ) |
| --- |

**In this phase, you’ll tailor the Streamlit page settings to suit your application’s needs. You’ll also define the title of your web page, which is displayed as the header in the browser tab and the main title on your app’s interface.**

**Session and messages store:**

| # Check for messages in session and create a title if not exists  if "messages" not in st.session\_state.keys():  st.session\_state.messages = [  {"role": "assistant", "content": "Hello, this is Gemini and how I can help you today?"}  ]  st.title(":rainbow[Howdy, How may I help you today?]")  # Display all messages for message in st.session\_state.messages:  with st.chat\_message(message["role"]):  st.write(message["content"]) |
| --- |

**This part of the code is responsible for managing the user’s session and storing the conversation messages. It ensures that the user’s interaction with the AI is persistent across the session, providing a seamless chat experience.**

**User input and storing in session:**

| # Receive user input user\_input = st.chat\_input()  # Store user input in session if user\_input is not None:  st.session\_state.messages.append({"role": "user", "content": user\_input})  with st.chat\_message("user"):  st.write(user\_input) |
| --- |

**This section deals with capturing the user’s input through the web interface. Once the user submits their message, the application will store it in the session, maintaining the flow of the conversation.**

**Generating AI response and displaying:**

| # Generate AI response and display if st.session\_state.messages[-1]["role"] != "assistant":  with st.chat\_message("assistant"):  with st.spinner("Loading..."):  ai\_response = model.generate\_content(user\_input)  st.write(ai\_response.text)  new\_ai\_message = {"role": "assistant", "content": ai\_response.text}  st.session\_state.messages.append(new\_ai\_message) |
| --- |

**Finally, the application will generate a response using the Gemini API based on the user’s input. The AI’s reply is then displayed in the chat interface, allowing the user to receive feedback or answers to their queries.**

### **Our Final Code would be like:**

| import streamlit as st import google.generativeai as genai  # Configure the Gemini API GOOGLE\_API\_KEY = 'ENTER\_YOUR\_API\_KEY\_HERE' genai.configure(api\_key=GOOGLE\_API\_KEY) model = genai.GenerativeModel('gemini-pro')  # Set Streamlit page configuration st.set\_page\_config(  page\_title="Gemini Clone",  page\_icon="✨",  layout="wide" )  # Check for messages in session and create a title if not exists if "messages" not in st.session\_state.keys():  st.session\_state.messages = [  {"role": "assistant", "content": "Hello, this is Gemini and how I can help you today?"}  ]  st.title(":rainbow[Howdy, How may I help you today?]")  # Display all messages for message in st.session\_state.messages:  with st.chat\_message(message["role"]):  st.write(message["content"]) # Receive user input user\_input = st.chat\_input()  # Store user input in session if user\_input is not None:  st.session\_state.messages.append({"role": "user", "content": user\_input})  with st.chat\_message("user"):  st.write(user\_input) # Generate AI response and display if st.session\_state.messages[-1]["role"] != "assistant":  with st.chat\_message("assistant"):  with st.spinner("Loading..."):  ai\_response = model.generate\_content(user\_input)  st.write(ai\_response.text)  new\_ai\_message = {"role": "assistant", "content": ai\_response.text}  st.session\_state.messages.append(new\_ai\_message) |
| --- |

### **Our Application would be like this:**