HauntScript: AI-Driven Horror Story Generator

🏗️ Architecture

HauntScript is designed using a modular architecture to ensure clarity, scalability, and ease of deployment. The architecture consists of four core components:

1. **Frontend (Streamlit):** The user interacts through a simple web interface where they input a character name, describe a horror scenario, and specify the number of lines for the story. Streamlit makes this possible with minimal code and quick deployment.

2. **Middleware (Application Logic):** This part handles processing the input, crafting prompts that are sent to the AI model, and capturing responses. It also includes exception handling to ensure the app remains responsive even during API errors.

3. **Backend (Gemini API):** The AI engine powering the horror story generation is Google's Gemini 1.5 Flash. It receives the crafted prompt, generates a horror story, and returns it to the middleware.

4. **Environment (Virtual Environment):** Everything runs inside a virtual environment named `horror_gen`. This isolates the project's dependencies from the system's Python installation and ensures version consistency.

Together, these components work seamlessly to generate unique horror stories on-demand.

⚙️ Prerequisites

To ensure smooth development and deployment of the *HauntScript* project, several essential tools and technologies were utilized. We chose **Python 3.10+** for compatibility with the latest libraries and better performance. **Anaconda** was used for creating and managing isolated virtual environments, which helped us avoid dependency conflicts and maintain consistent development across machines.

The core user interface was built using **Streamlit**, a lightweight framework that allowed us to deploy a fully functional web application with minimal effort. To connect our app with Google's Gemini language model, we used the **Google Generative AI SDK**, which enabled seamless integration with pre-trained LLMs. **Visual Studio Code** served as our integrated development environment, supporting efficient code editing, debugging, and package management.

🔧 Tools and Technologies Used

- ✅ Python 3.10+
- ✅ Anaconda (for environment management)
- ✅ Streamlit (for the web UI)
- ✅ Google Generative AI SDK (to interact with Gemini)
- ✅ Visual Studio Code (as the development environment)

To install the required packages, we ran:

```
pip install streamlit google-generativeai
```

These dependencies were also saved in a `requirements.txt` file for easier setup and sharing across systems.

📁 Project Structure

The project followed a minimal and well-organized folder structure to ensure clarity and ease of navigation. Each file serves a distinct and important purpose, keeping the implementation simple and efficient.

Directory Layout:

```
├── horror.py            → Contains the main Streamlit application code including the UI and logic to interact with the Gemini model

├── requirements.txt     → Lists all the necessary Python packages and dependencies required to run the project
```

- **`horror.py`**: The central script of the application. It handles user input, generates prompts, communicates with the Gemini API, and displays the output.

- `requirements.txt`: This file contains a list of all libraries and tools that must be installed in the virtual environment to ensure the application runs without issues.

🧩 Milestone 1: Requirements Specification

We outlined the following:

Functional Requirements
- Accept user input: name, situation, and number of lines.
- Connect to the Gemini model and return a story.
- Display story in a neat format in the browser.

Non-Functional Requirements
- Must return a response within 3–5 seconds.
- Must run without dependency conflicts.
- UI must be intuitive and visually aligned with the horror theme.
- Must support future updates with ease.

🧠 Initializing The Model

We initialized the Gemini model using the following parameters for best performance in creative generation tasks:

python

```
api_key = "<your_actual_api_key_here>"
genai.configure(api_key=api_key)

model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config={
        "temperature": 0.75,
        "top_p": 0.95,
        "top_k": 64,
        "max_output_tokens": 8192,
        "response_mime_type": "text/plain",
    }
)
```

These parameters make the story generation slightly creative while staying within the context.

🤖 Interfacing With Pre-Trained Model

At the heart of the *HauntScript* application lies the function responsible for crafting the horror story. This function dynamically builds a prompt based on the user's input and communicates with the Gemini model to generate the story.

Function Overview:

python

```python
def generate_horror_story(character_name, situation, no_of_lines):

    prompt = f"Write me a horror story with the character name '{character_name}' and situation '{situation}' in {no_of_lines} lines."

    chat_session = model.start_chat(history=[])

    response = chat_session.send_message(prompt)

    return response.text
```

Explanation:

- 📌 `character_name`, `situation`, and `no_of_lines` are provided by the user through the Streamlit UI.
- 📌 A prompt is dynamically constructed using these inputs.
- 📌 `start_chat(history=[])` initializes a new chat session with the Gemini model.
- 📌 The prompt is sent using `send_message()`, and the model's response is returned.

.

🚀 Model Deployment

To run the **HauntScript** application locally, follow the steps below to set up the virtual environment, install dependencies, and launch the Streamlit app.

---

1. Open Anaconda Prompt

You can find it in the Start Menu (Windows) or launch it from your system.

---

2. Create and Activate the Virtual Environment

Run the following commands to create a new environment named `horror_gen` with Python 3.10 and activate it:

conda create --name horror_gen python=3.10

conda activate horror_gen

---

3. Install Required Dependencies

After activating the environment, install the necessary Python packages:

pip install streamlit google-generativeai

---

4. Run the Streamlit Application

Use the command below to launch the app:

streamlit run horror.py

---

5. Access the App in Your Browser

Once launched, the app will automatically open in your default browser. If not, visit the following URL manually:

http://localhost:8501

This deployment approach ensures a clean, isolated environment and a seamless user experience.

🔐 Google Gemini API Integration

We created our API key on Google AI Studio by enabling the Gemini API. The key was securely embedded into our Streamlit app.

Best practice suggests storing API keys in environment variables and loading them using `os.environ` in Python to prevent exposure in source code.

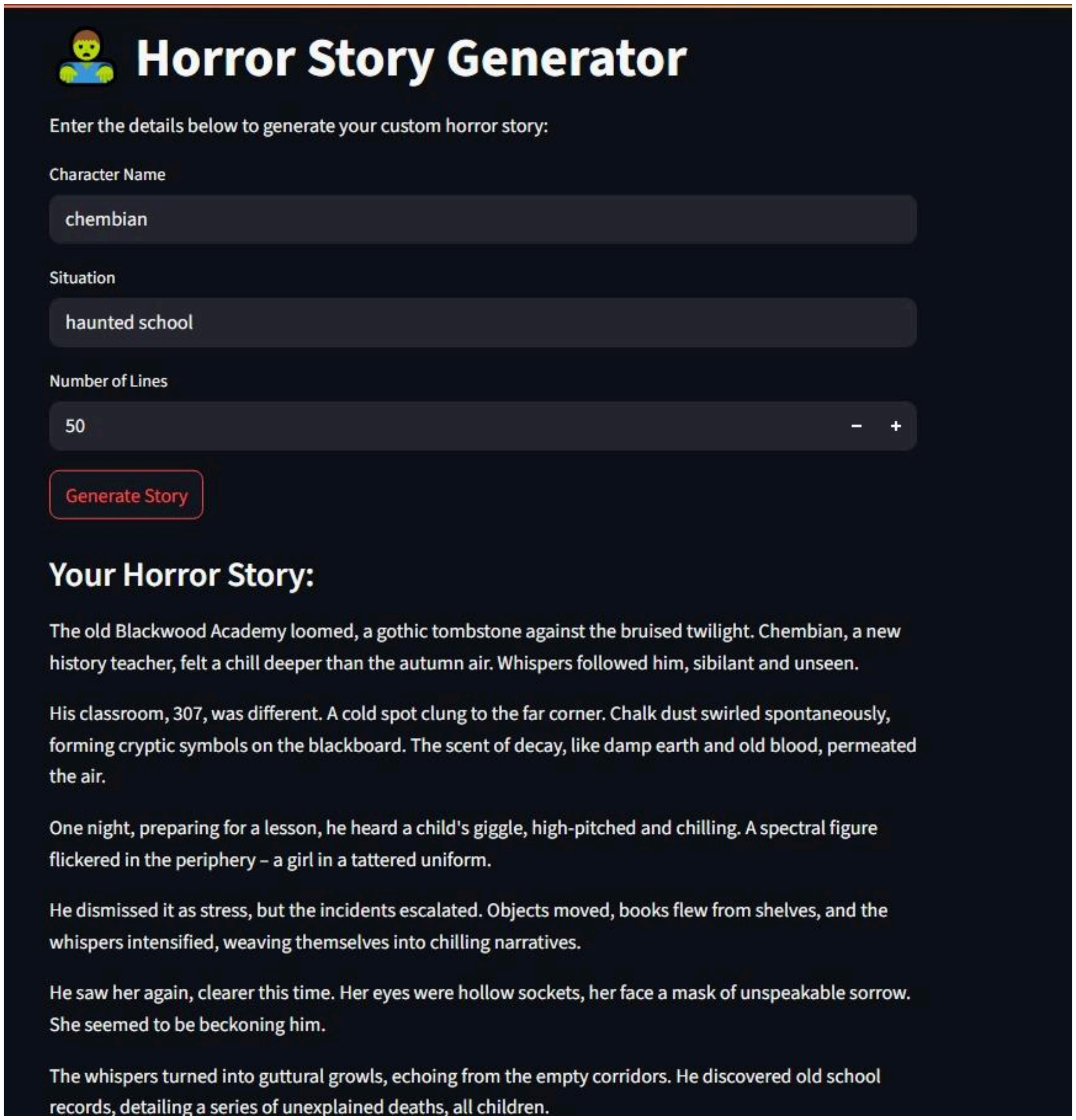
🧪 Sample Code Walkthrough

The `horror.py` file contains:

- UI setup using `st.text_input()` and `st.number_input()`
- Function call to `generate_horror_story(...)`
- Output rendered using `st.write()`

This modular code was easy to debug and scale.

🧾 Sample Output



🧟 Horror Story Generator

Enter the details below to generate your custom horror story:

Character Name: chembian

Situation: haunted school

Number of Lines: 50

Your Horror Story:

The old Blackwood Academy loomed, a gothic tombstone against the bruised twilight. Chembian, a new history teacher, felt a chill deeper than the autumn air. Whispers followed him, sibilant and unseen.

His classroom, 307, was different. A cold spot clung to the far corner. Chalk dust swirled spontaneously, forming cryptic symbols on the blackboard. The scent of decay, like damp earth and old blood, permeated the air.

One night, preparing for a lesson, he heard a child's giggle, high-pitched and chilling. A spectral figure flickered in the periphery – a girl in a tattered uniform.

He dismissed it as stress, but the incidents escalated. Objects moved, books flew from shelves, and the whispers intensified, weaving themselves into chilling narratives.

He saw her again, clearer this time. Her eyes were hollow sockets, her face a mask of unspeakable sorrow. She seemed to be beckoning him.

The whispers turned into guttural growls, echoing from the empty corridors. He discovered old school records, detailing a series of unexplained deaths, all children.

🔭 Future Scope

Ideas for further improvement:

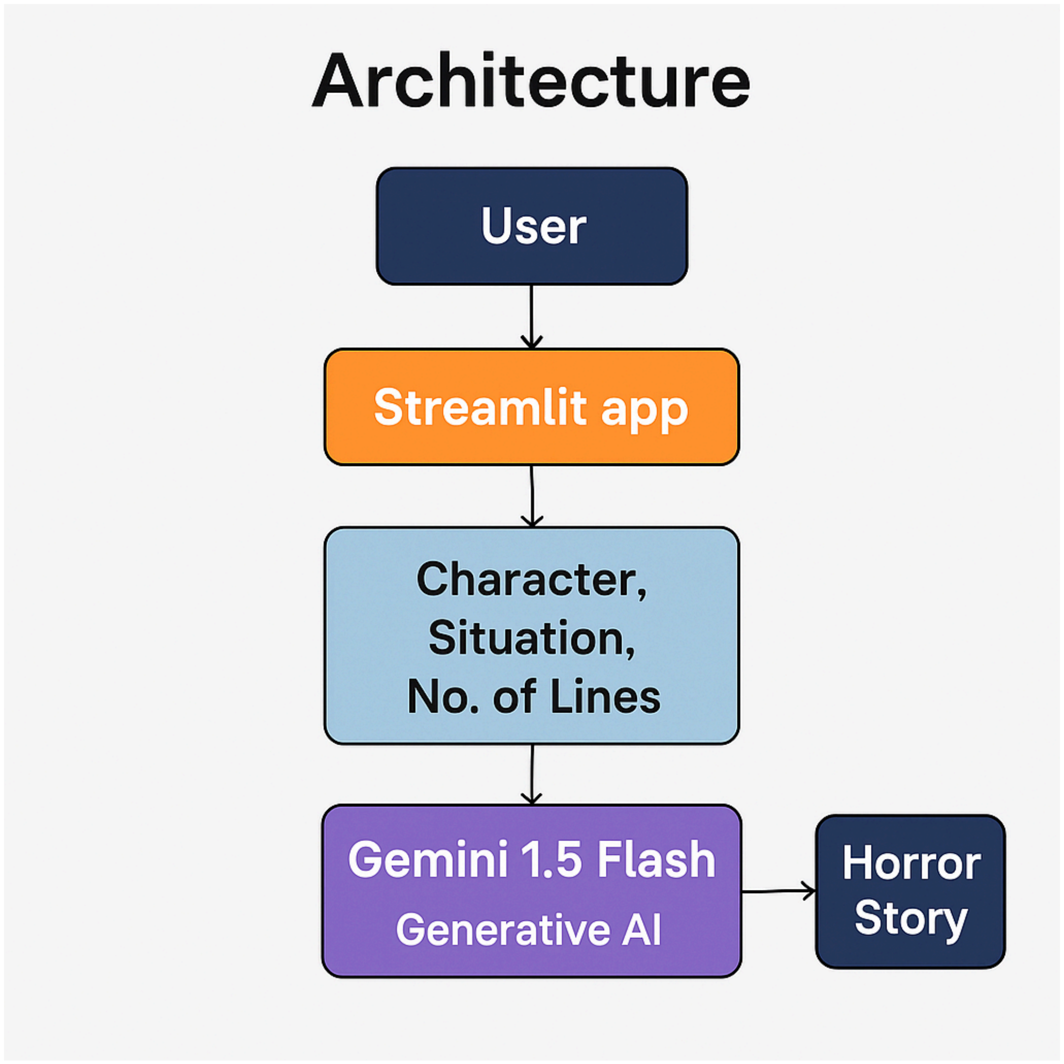
- Add options to save/export story as PDF
- Add horror audio effects
- Allow genre-switching (sci-fi, thriller, fantasy)
- Host the application online (Streamlit Cloud or Render)
- Let users sign in and view history

✅ Conclusion

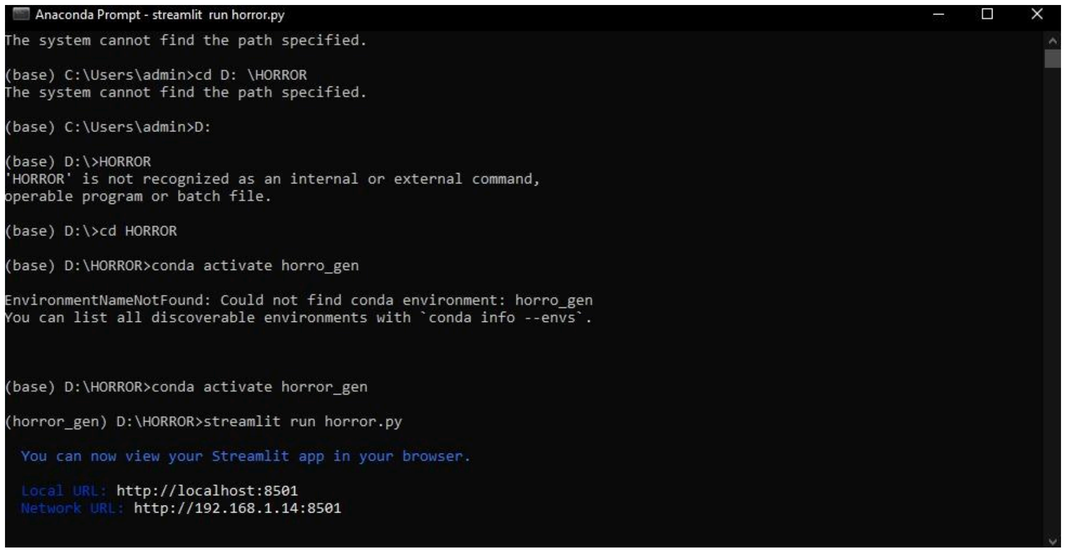
HauntScript was built as a creative fusion of generative AI and storytelling. Using Google Gemini with a Streamlit interface made development smooth and results visually appealing.

This project shows the potential of LLMs in personalized story generation and sets a solid foundation for future interactive fiction apps.

Architecture Diagram



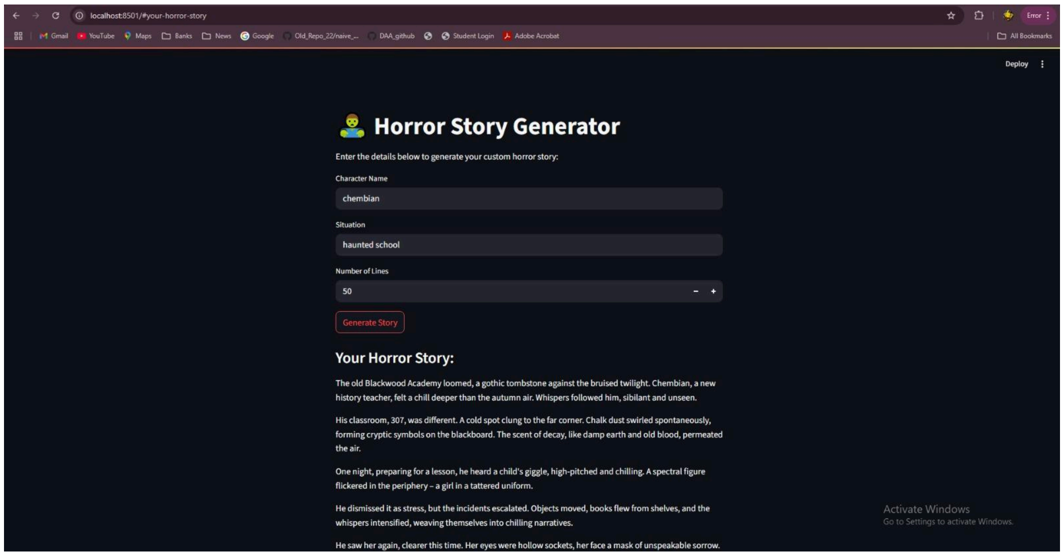
Anaconda Prompt



Anaconda Prompt - streamlit run horror.py

```
The system cannot find the path specified.

(base) C:\Users\admin>cd D: \HORROR
The system cannot find the path specified.

(base) C:\Users\admin>D:

(base) D:\>HORROR
'HORROR' is not recognized as an internal or external command,
operable program or batch file.

(base) D:\>cd HORROR

(base) D:\HORROR>conda activate horro_gen

EnvironmentNameNotFound: Could not find conda environment: horro_gen
You can list all discoverable environments with `conda info --envs`.



(base) D:\HORROR>conda activate horror_gen

(horror_gen) D:\HORROR>streamlit run horror.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.14:8501
```

Streamlit App Screenshot



Horror Story Generator

Enter the details below to generate your custom horror story:

Character Name: chembian

Situation: haunted school

Number of Lines: 50

Generate Story

Your Horror Story:

The old Blackwood Academy loomed, a gothic tombstone against the bruised twilight. Chembian, a new history teacher, felt a chill deeper than the autumn air. Whispers followed him, sibilant and unseen.

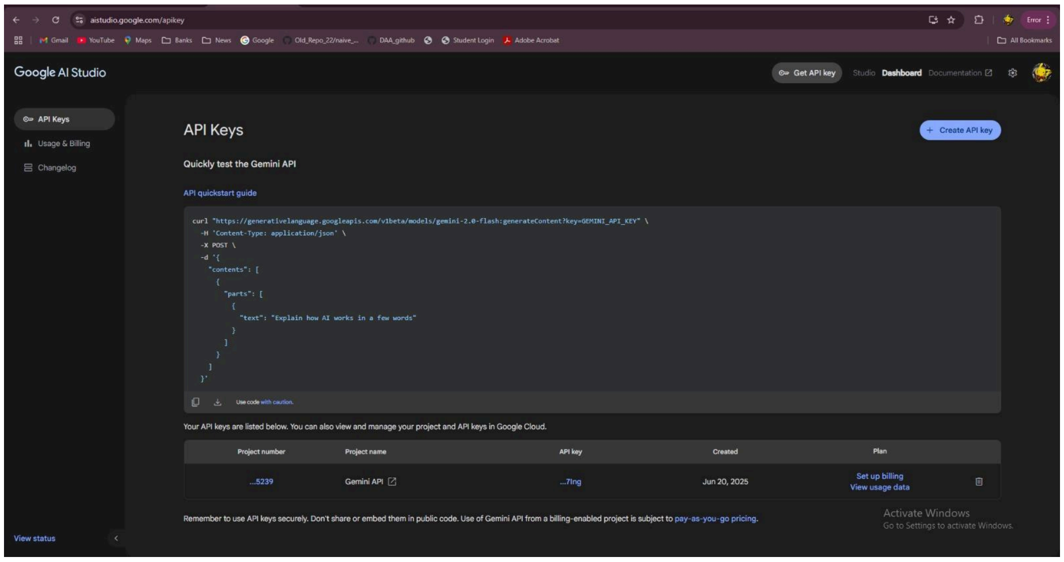
His classroom, 307, was different. A cold spot clung to the far corner. Chalk dust swirled spontaneously, forming cryptic symbols on the blackboard. The scent of decay, like damp earth and old blood, permeated the air.

One night, preparing for a lesson, he heard a child's giggle, high-pitched and chilling. A spectral figure flickered in the periphery – a girl in a tattered uniform.

He dismissed it as stress, but the incidents escalated. Objects moved, books flew from shelves, and the whispers intensified, weaving themselves into chilling narratives.

He saw her again, clearer this time. Her eyes were hollow sockets, her face a mask of unspeakable sorrow.

Google Gemini API Page



API Keys

Quickly test the Gemini API

API quickstart guide

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=GEMINI_API_KEY" \
  -H 'Content-Type: application/json' \
  -X POST \
  -d '{
    "contents": [
      {
        "parts": [
          {
            "text": "Explain how AI works in a few words"
          }
        ]
      }
    ]
  }'
```

Use code with caution.

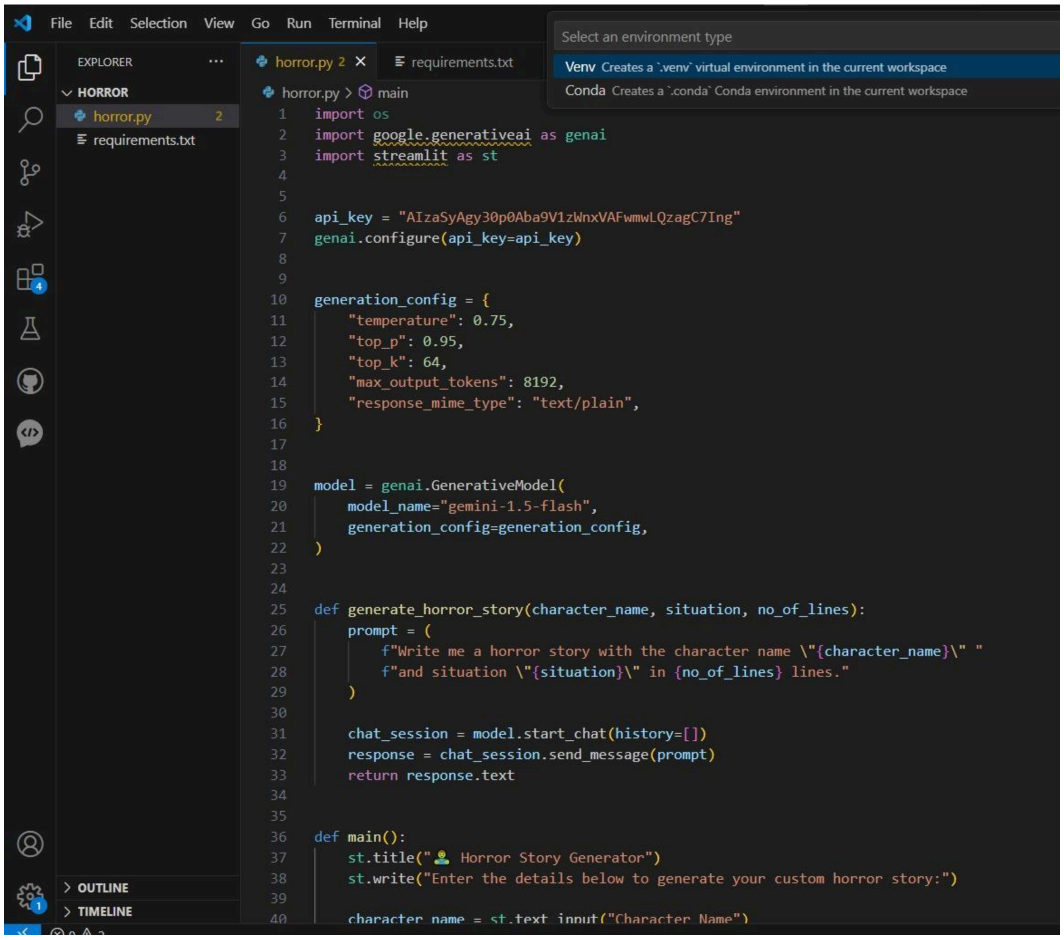
Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.

Project number	Project name	API key	Created	Plan
...5239	Gemini API	...7Ing	Jun 20, 2025	Set up billing View usage data

Remember to use API keys securely. Don't share or embed them in public code. Use of Gemini API from a billing-enabled project is subject to pay-as-you-go pricing.

Activate Windows
Go to Settings to activate Windows.

Code Part 1



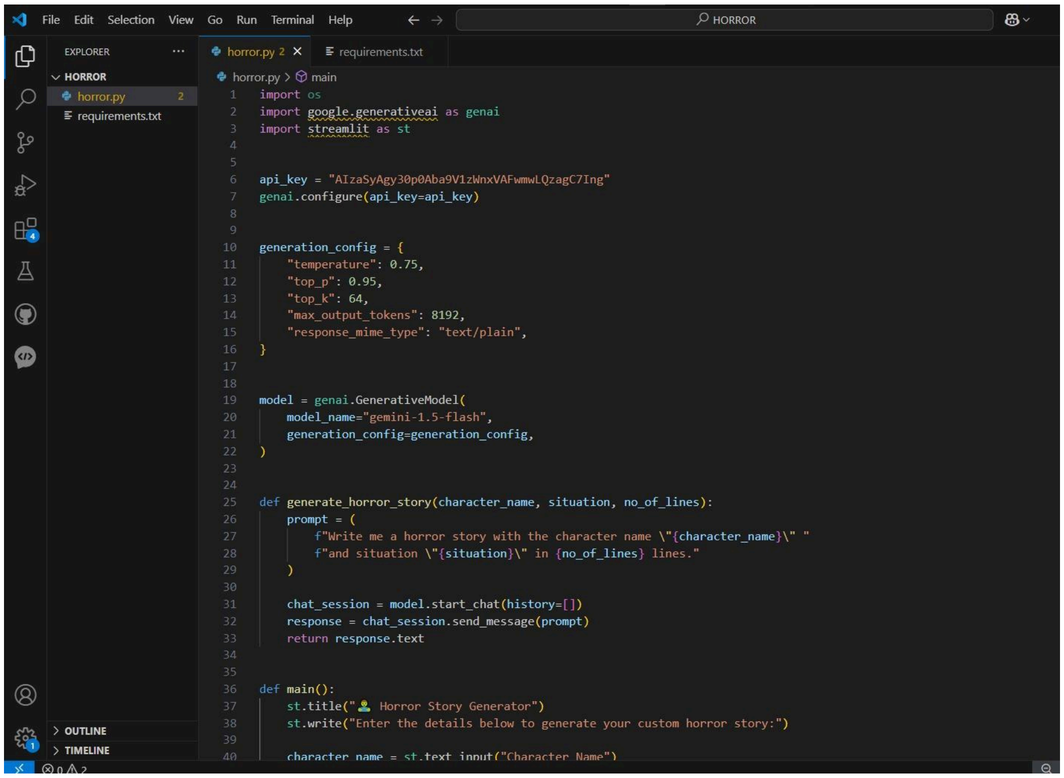
Select an environment type

Venv Creates a `.venv` virtual environment in the current workspace

Conda Creates a `.conda` Conda environment in the current workspace

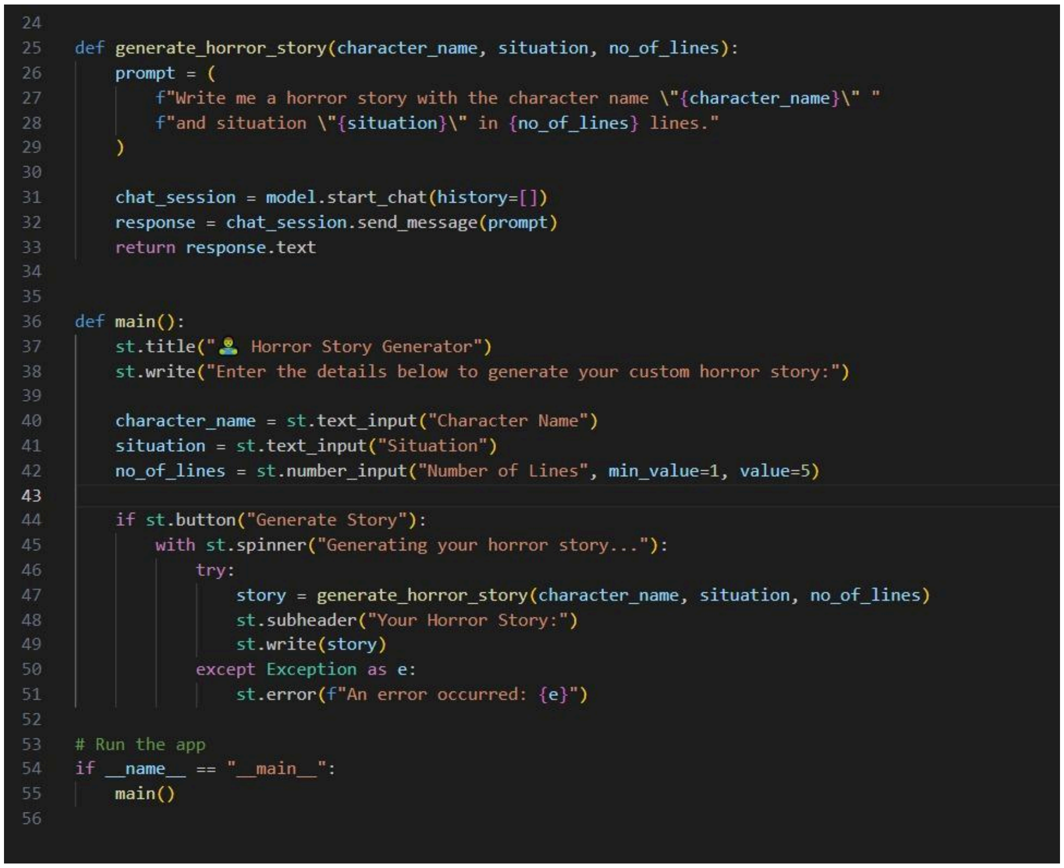
```python
import os
import google.generativeai as genai
import streamlit as st


api_key = "AIzaSyAgy30p0Aba9V1zWnxVAFwmwLQzagC7Ing"
genai.configure(api_key=api_key)


generation_config = {
    "temperature": 0.75,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}


model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config=generation_config,
)


def generate_horror_story(character_name, situation, no_of_lines):
    prompt = (
        f"Write me a horror story with the character name \"{character_name}\" "
        f"and situation \"{situation}\" in {no_of_lines} lines."
    )

    chat_session = model.start_chat(history=[])
    response = chat_session.send_message(prompt)
    return response.text


def main():
    st.title("🧟 Horror Story Generator")
    st.write("Enter the details below to generate your custom horror story:")

    character_name = st.text_input("Character Name")
```

Code Part 2



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


api_key = "AIzaSyAgy30p0Aba9V1zWnxVAFwmwLQzagC7Ing"
genai.configure(api_key=api_key)


generation_config = {
    "temperature": 0.75,
    "top_p": 0.95,
    "top_k": 64,
    "max_output_tokens": 8192,
    "response_mime_type": "text/plain",
}


model = genai.GenerativeModel(
    model_name="gemini-1.5-flash",
    generation_config=generation_config,
)


def generate_horror_story(character_name, situation, no_of_lines):
    prompt = (
        f"Write me a horror story with the character name \"{character_name}\" "
        f"and situation \"{situation}\" in {no_of_lines} lines."
    )

    chat_session = model.start_chat(history=[])
    response = chat_session.send_message(prompt)
    return response.text


def main():
    st.title("🧟 Horror Story Generator")
    st.write("Enter the details below to generate your custom horror story:")

    character_name = st.text_input("Character Name")
```

Code Part 3



```python

def generate_horror_story(character_name, situation, no_of_lines):
    prompt = (
        f"Write me a horror story with the character name \"{character_name}\" "
        f"and situation \"{situation}\" in {no_of_lines} lines."
    )

    chat_session = model.start_chat(history=[])
    response = chat_session.send_message(prompt)
    return response.text


def main():
    st.title("🧟 Horror Story Generator")
    st.write("Enter the details below to generate your custom horror story:")

    character_name = st.text_input("Character Name")
    situation = st.text_input("Situation")
    no_of_lines = st.number_input("Number of Lines", min_value=1, value=5)

    if st.button("Generate Story"):
        with st.spinner("Generating your horror story..."):
            try:
                story = generate_horror_story(character_name, situation, no_of_lines)
                st.subheader("Your Horror Story:")
                st.write(story)
            except Exception as e:
                st.error(f"An error occurred: {e}")

# Run the app
if __name__ == "__main__":
    main()

```