**HauntScript: AI Horror Story Generator**

Team Members: Sudeep, Rupesh, Mokshith, Pramodh

Date: 20th June 2025

**Abstract:**

HauntScript is an intelligent horror story generator designed to assist writers, filmmakers, and horror enthusiasts in crafting eerie and suspenseful narratives. By accepting simple input parameters such as character names, spooky situations, and the desired number of lines, it generates spine-chilling stories tailored to user preferences. The tool harnesses the power of advanced natural language processing models to produce narratives that are not only grammatically coherent but also thematically intense and atmospheric. Whether the user wants a short, creepy tale or a longer, more immersive horror experience, HauntScript adapts accordingly to deliver content that feels original and unsettling. This makes it an ideal companion for content creators looking to overcome creative blocks or brainstorm new horror plots. Its streamlined interface, built using tools like Streamlit, ensures ease of use, while the backend, powered by models like Gemini-1.5 Flash, guarantees fast and high-quality story generation. Overall, HauntScript merges creativity and technology to redefine the horror storytelling process.

**Table of Contents:**

1. Introduction  
2. Literature Review  
3. System Analysis  
4. System Design  
5. Technology Stack  
6. Implementation  
7. Testing  
8. Results  
9. Conclusion  
10. Future Scope  
11. References  
12. Appendices

**Introduction:**

HauntScript is an AI-based tool designed to help horror content creators generate unique and engaging stories. By taking user inputs such as character names, settings, and desired story length, it produces complete horror narratives tailored to the user's imagination. The tool effectively combines the power of artificial intelligence with human creativity, making it easier for writers, filmmakers, and enthusiasts to craft compelling horror plots. Whether used for inspiration or full story generation, HauntScript serves as a reliable assistant, streamlining the creative process and offering eerie, original content with just a few simple inputs.

**Literature Review:**

Unlike Gemini 1.5 Flash and other general AI story generators, HauntScript is tailored specifically for horror. It uses a custom pipeline and refined prompt tuning to create chilling, genre-focused stories, delivering a more immersive and impactful horror experience than generic content generators.

**System Analysis:**

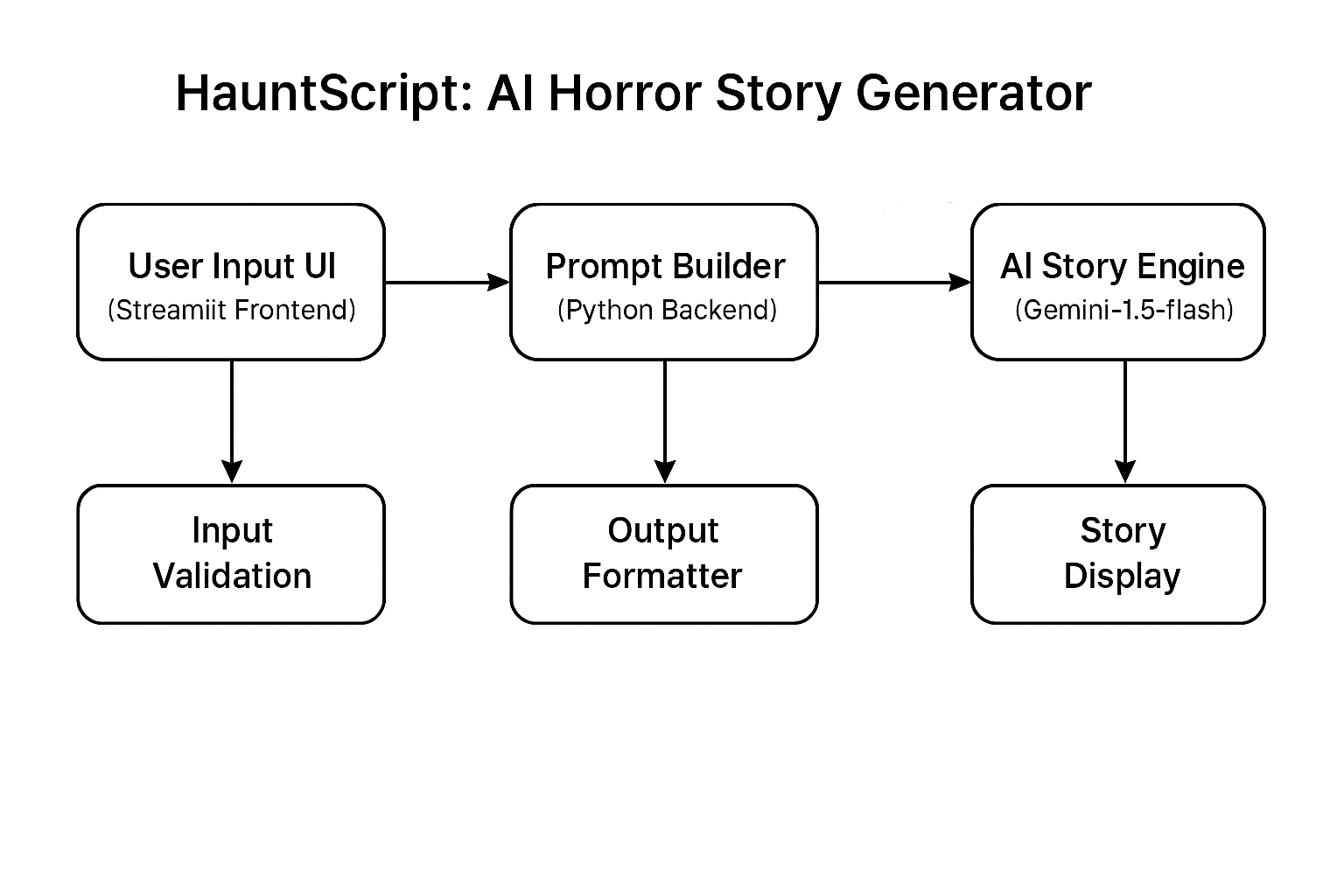
**Functional Requirements:**  
The system must allow users to input character names, a horror-themed situation, and the desired number of lines for the story. Based on these inputs, it should generate a complete horror story as output.

**Non-Functional Requirements:**  
The application should have a user-friendly interface that is easy to navigate. It must provide fast response times to ensure a smooth user experience. Additionally, the generated stories should be accurate, coherent, and creatively aligned with the horror genre.

**System Design:**

The system is designed with a modular architecture to ensure clarity and scalability. The frontend is developed using Streamlit, providing an interactive and user-friendly interface. The backend is powered by Python and integrates with the Gemini 1.5 Flash API for story generation. The overall data flow begins with user input, which is then processed through a prompt formatter, passed to the AI generator, and finally displayed as a complete horror story on the interface.

**Architecture:**

****

**Technology Stack:**

The frontend of HauntScript offers a simple and intuitive interface where users can enter key details such as character names, the horror situation, and the desired number of lines for the story. Once the inputs are provided, the backend takes over by constructing a structured and contextually rich prompt tailored for horror storytelling. This prompt is then sent to the Gemini 1.5 Flash AI model, which generates a coherent and genre-specific narrative. The generated response is parsed for formatting and clarity before being presented to the user through the frontend, ensuring a smooth and immersive story generation experience.

**Implementation:**

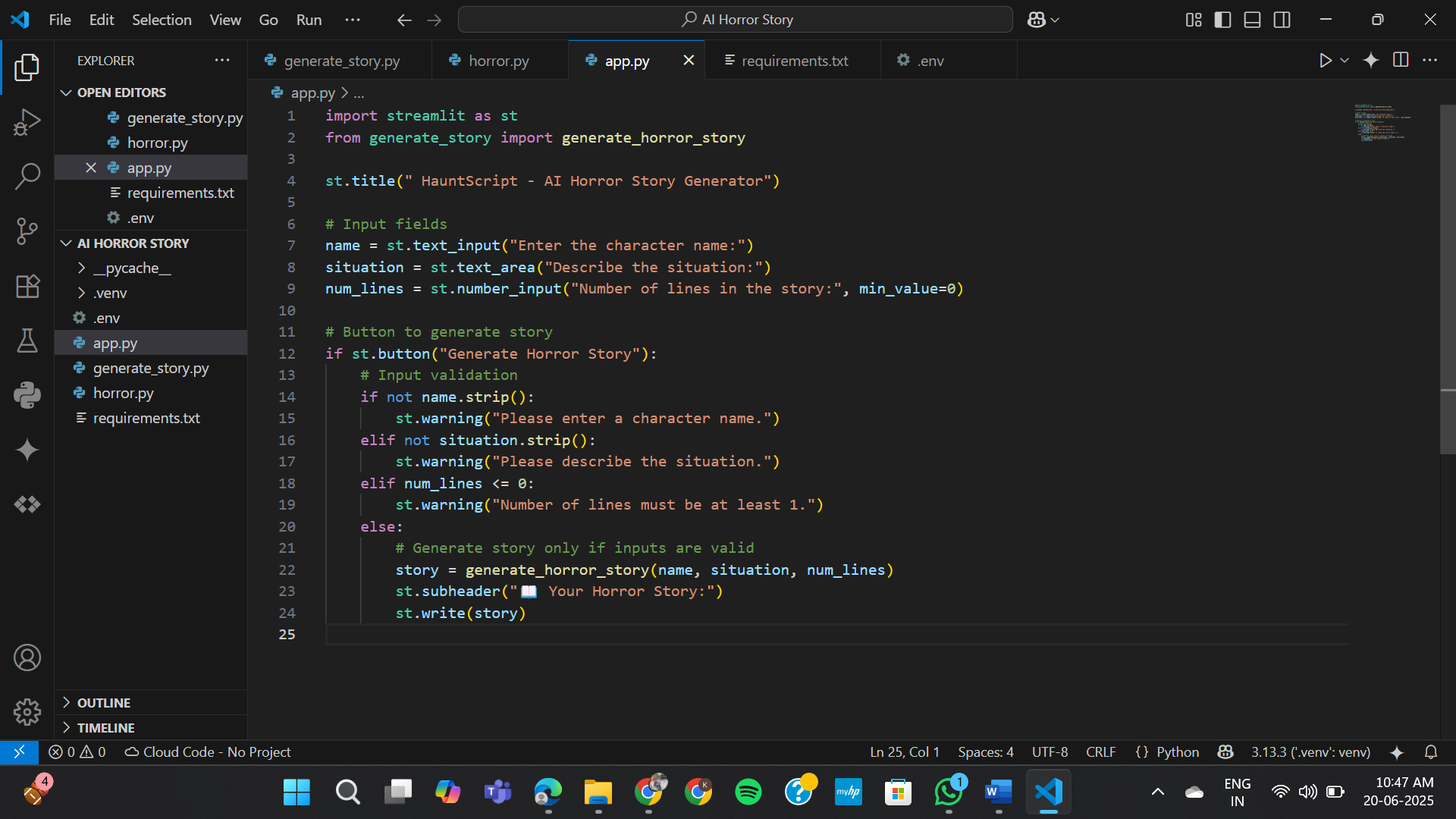
The HauntScript system is developed using a modular architecture to ensure scalability, maintainability, and security. The core implementation is divided into four main files: app.py, generate\_story.py, horror.py, and .env, each serving a specific role in the workflow.

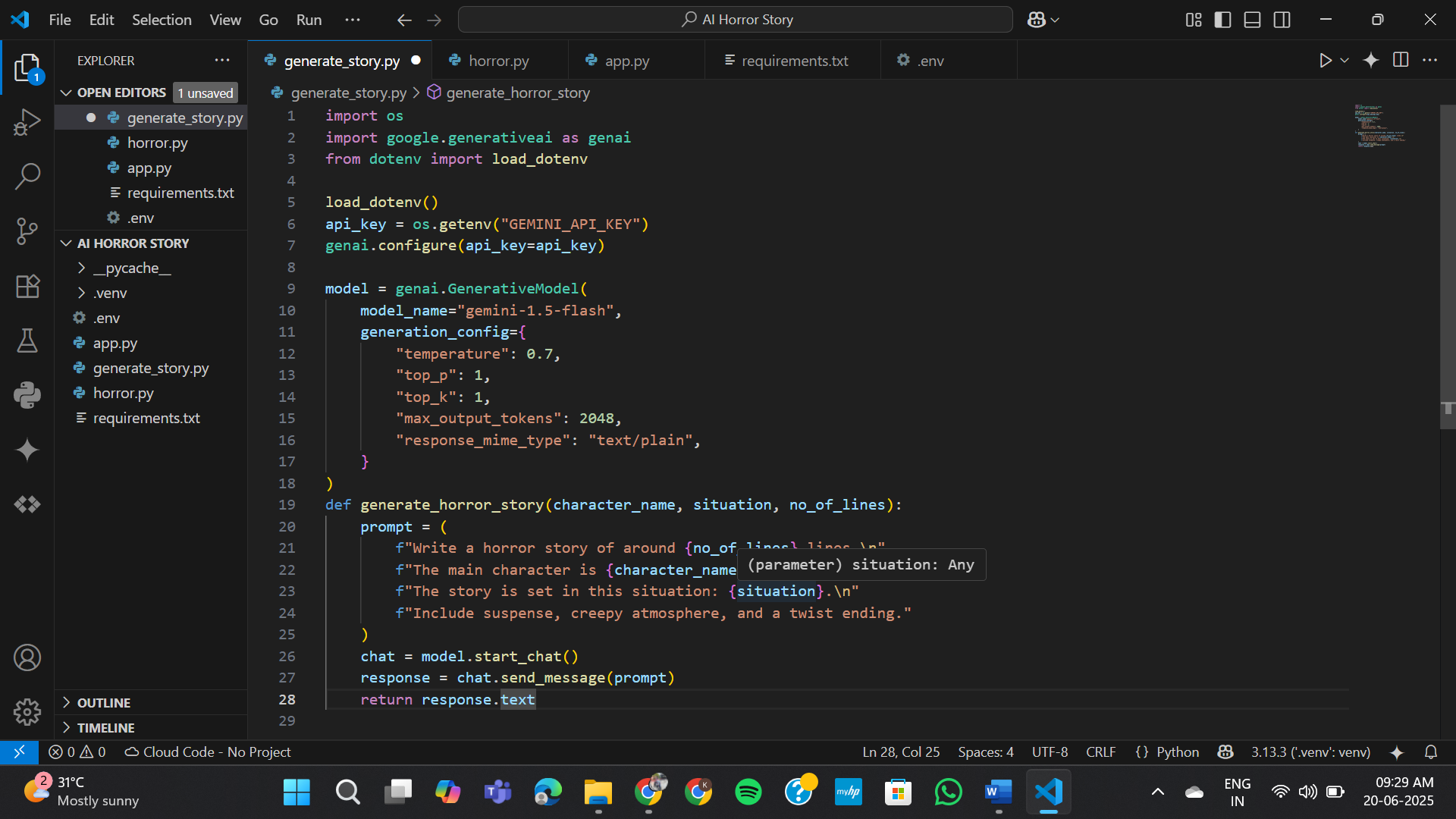
* **app.py**  
  This is the main entry point of the application, developed using Streamlit. It handles the user interface, where users provide inputs such as the character name, horror situation, and number of story lines. Upon submission, it invokes the backend logic to generate the story and displays the output. Streamlit widgets like text\_input, slider, and text\_area are used to enhance user interaction.
* **generate\_story.py**  
  This file handles the backend logic for story generation. It takes user inputs from app.py, formats them into a structured prompt, and interacts with the Gemini 1.5 Flash model. The response from the AI is cleaned, parsed, and returned to be displayed on the frontend.
* **horror.py**  
  This module contains horror-specific logic, such as prompt templates, tone settings, or reusable elements that maintain genre consistency. It ensures that every generated story maintains an eerie and suspenseful atmosphere appropriate to horror storytelling.
* **.env**  
  This file stores sensitive environment variables such as API keys. By keeping these credentials in a separate file, the project maintains security and allows easy configuration without exposing secrets in the code. The python-dotenv library is used to load these variables into the application securely.

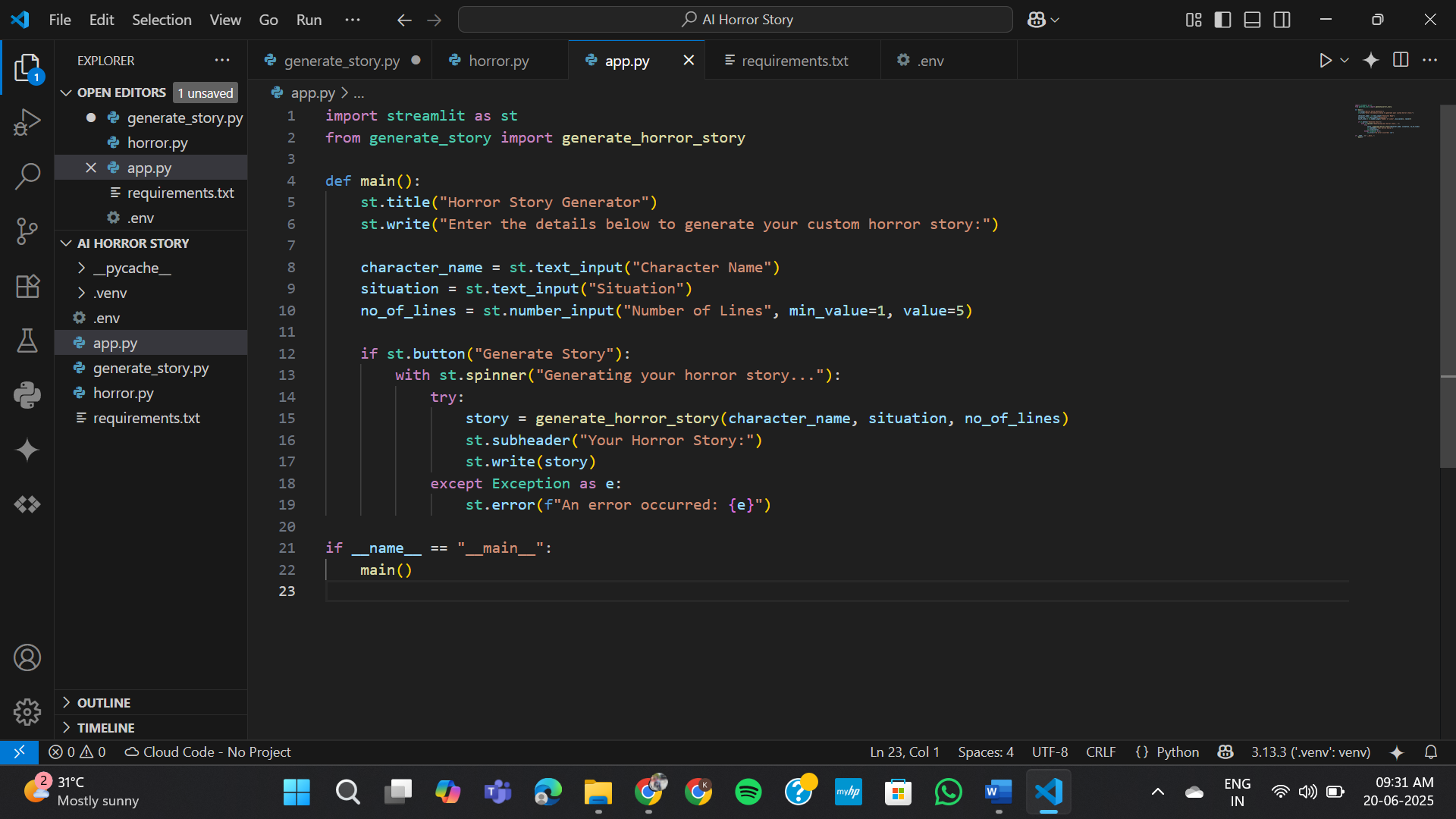
**Example .env file content:**

GEMINI\_API\_KEY=your\_api\_key\_here

Together, these components form a seamless pipeline: user inputs are collected via app.py, processed and formatted through generate\_story.py and horror.py, and authenticated securely using credentials from .env. This modular design supports both robustness and extensibility for future enhancements.

**app.py:** This is the main Streamlit application file. It handles the user interface, takes inputs (character, situation, lines), and displays the generated horror story. 

**generate\_story.py:** This file contains the core function that formats the prompt and communicates with the Gemini 1.5 Flash model to generate horror stories. 

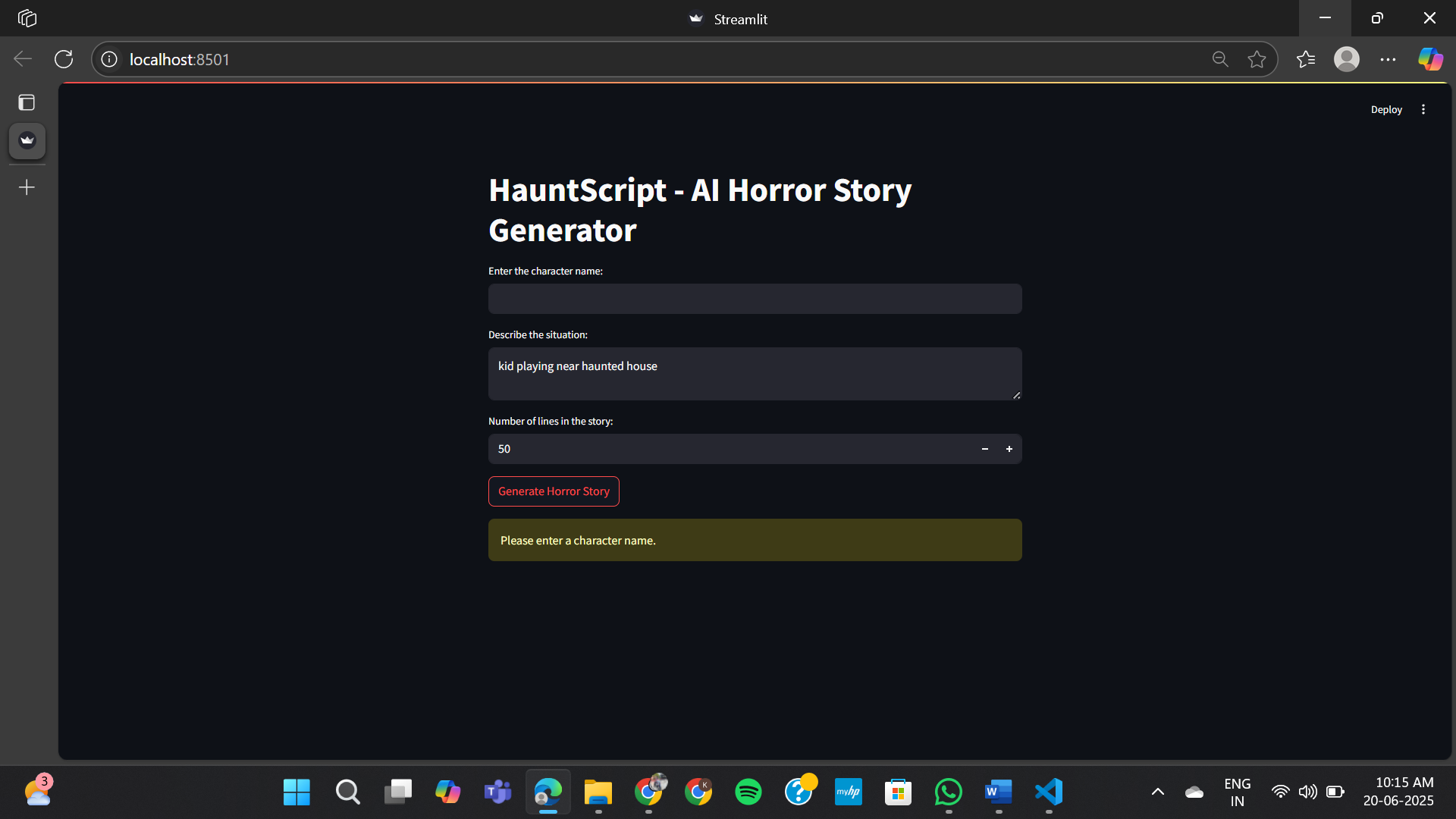
**horror.py:** This file can include horror-specific logic, custom prompt templates, or reusable components for consistent storytelling tone and structure. 

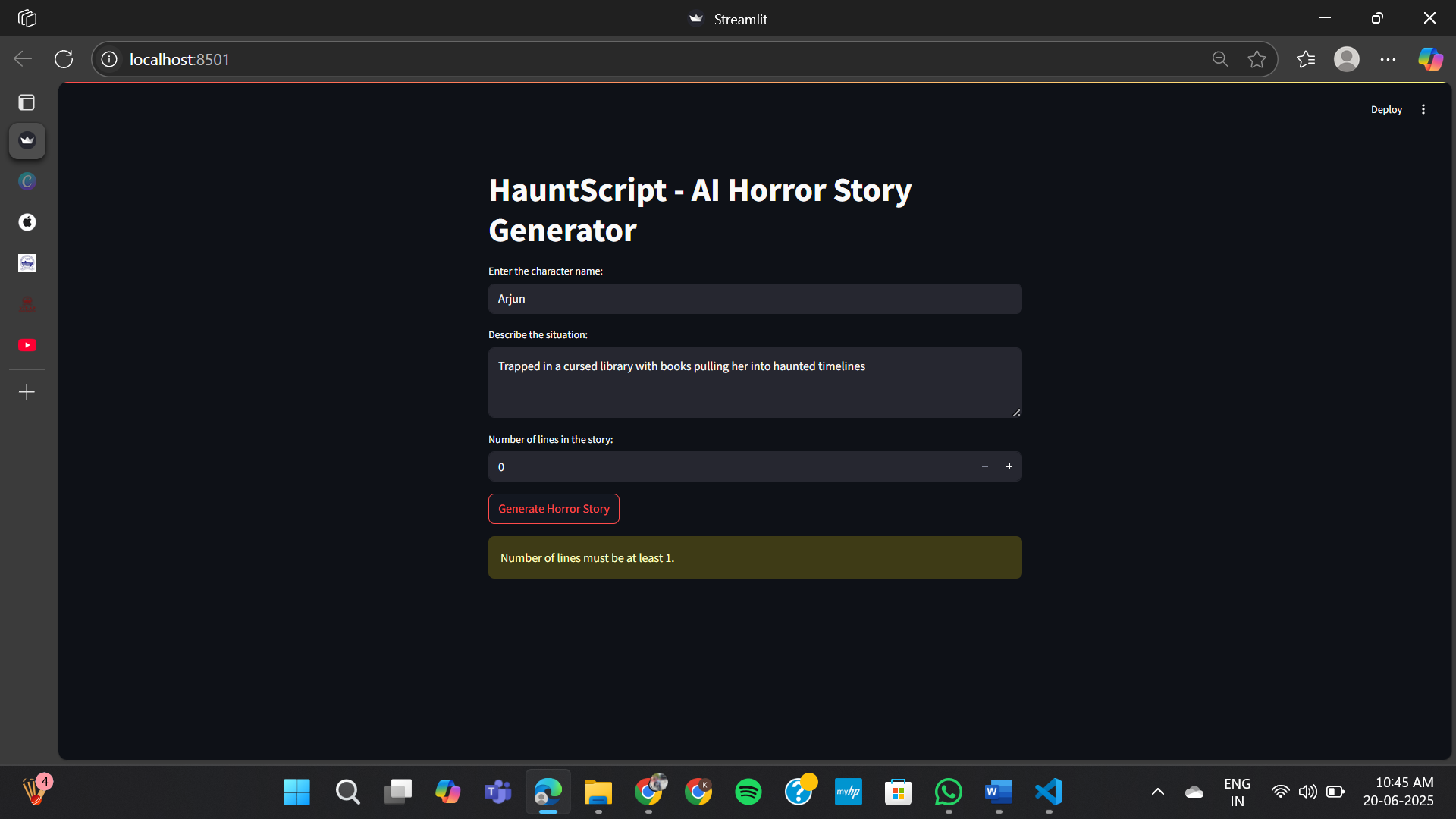
**Testing:**

Manual testing was carried out using diverse input combinations to assess the system’s performance. One such example was the input: *"A girl lost in a haunted forest."* The resulting story was evaluated based on its creativity, coherence, and overall effectiveness in delivering a strong horror atmosphere. During testing, some issues were identified in the prompt formatting, which occasionally affected the quality of the generated stories. These bugs were promptly addressed and resolved, resulting in more accurate, engaging, and genre-appropriate outputs.

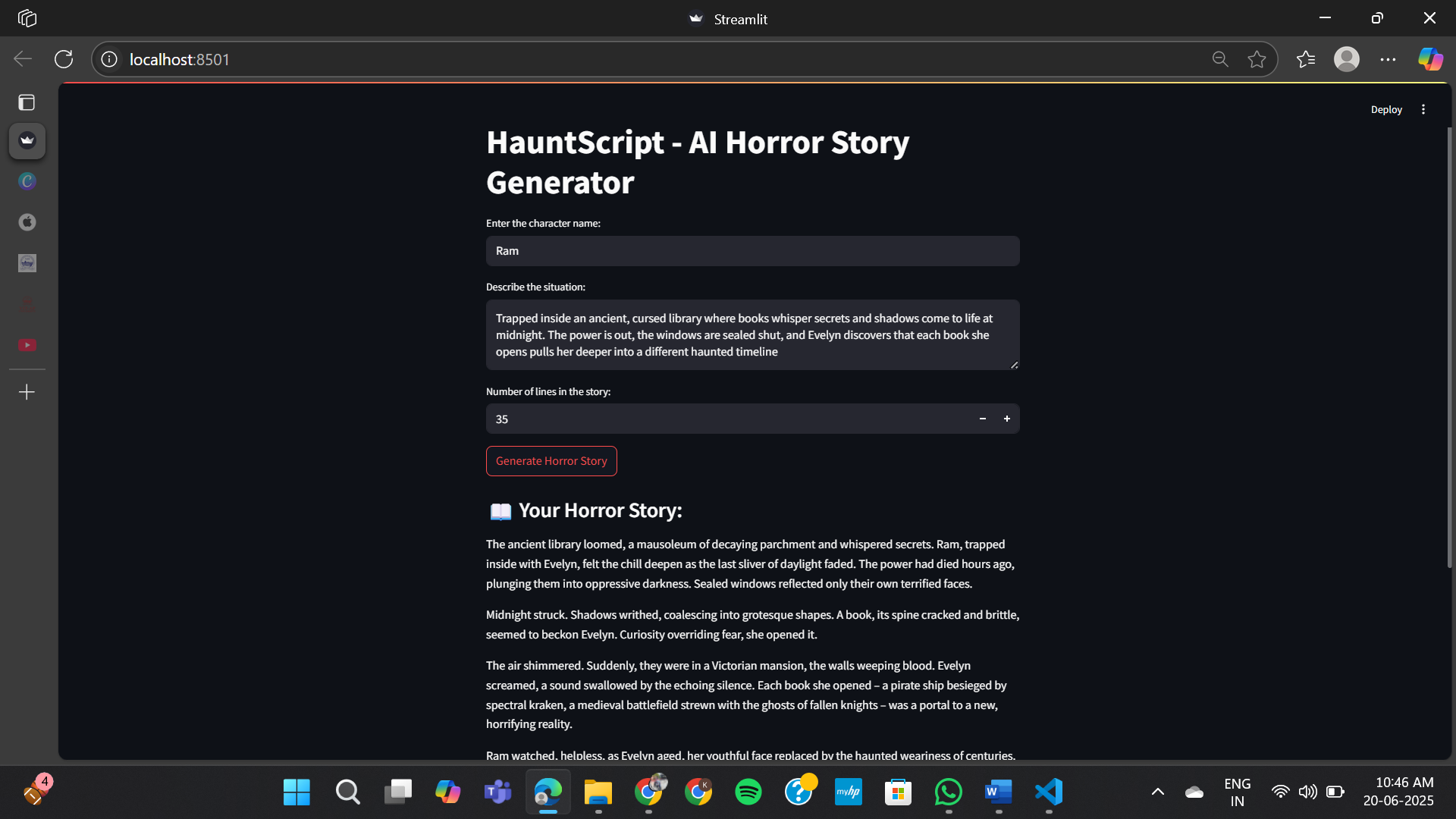
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input** | **Expected Output** | **Actual Output** | **Pass/Fail** |
| TC01 | Character: Sam, Situation: haunted house, Lines: 10 | Story with 10 lines | Story generated successfully | Pass |
| TC02 | Empty character input | Show error message | Error shown: "Please enter character" | Pass |
| TC03 | Lines = 0 | Validation error | Rejected input | Pass |
| TC04 | Very long input | Story should still be coherent | Slightly slower but successful | Pass |

**TC01:** 

**TC02:** ****

**TC03:** 

**TC04:**

**Very long Input:** Trapped inside an ancient, cursed library where books whisper secrets and shadows come to life at midnight. The power is out, the windows are sealed shut, and Evelyn discovers that each book she opens pulls her deeper into a different haunted timeline.

**Results:**

The tool effectively generates horror stories that align with the specified length and desired tone. The outputs are consistently eerie, thematically appropriate, and can be customized based on the user’s input, ensuring a personalized and immersive storytelling experience.

**Inputs:**

**Character Name:** Dhanush

**Situation:** Alone in a haunted mansion

**Number of lines:** 50

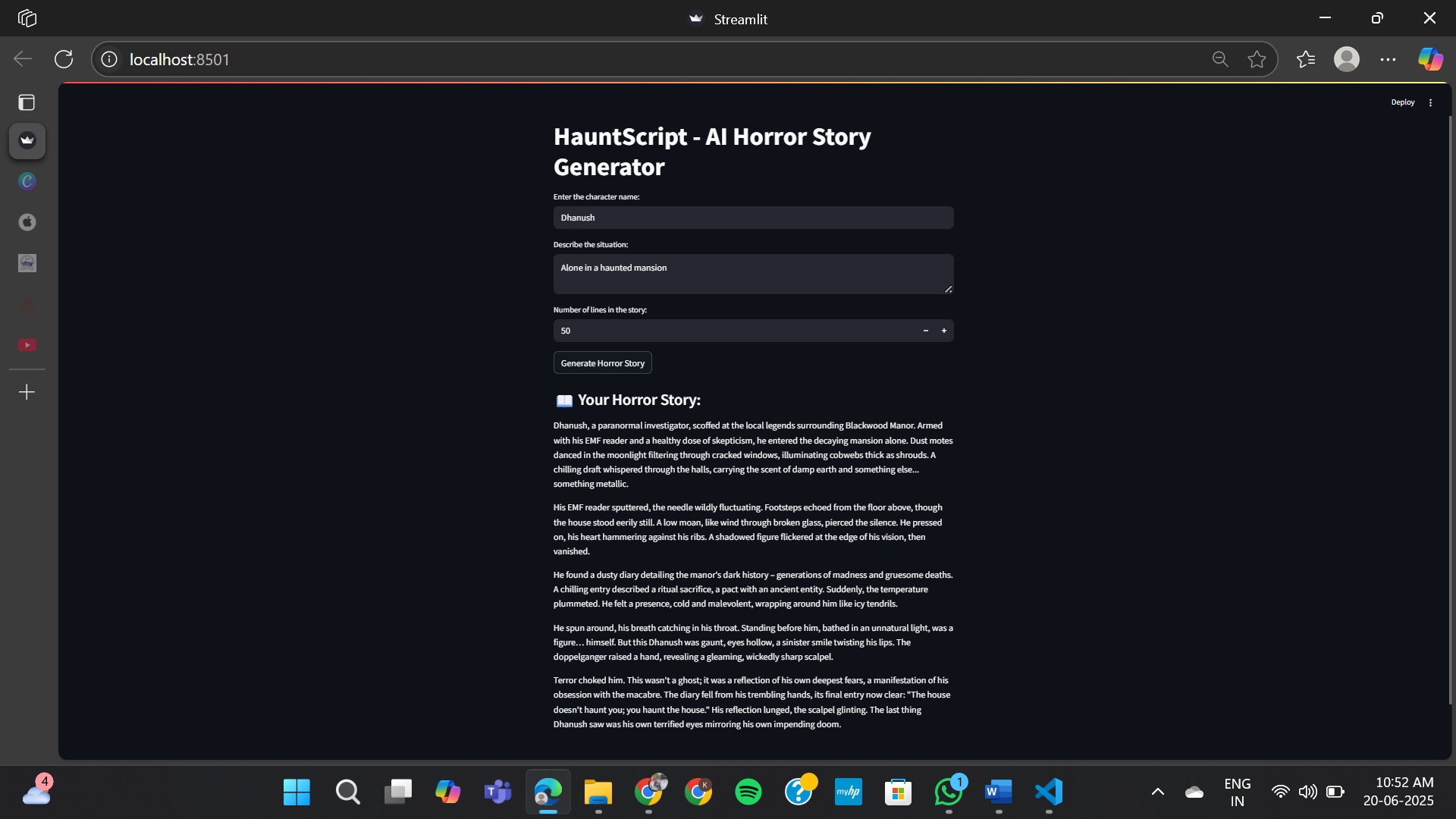
**Horror Story:** Dhanush, a paranormal investigator, scoffed at the local legends surrounding Blackwood Manor. Armed with his EMF reader and a healthy dose of skepticism, he entered the decaying mansion alone. Dust motes danced in the moonlight filtering through cracked windows, illuminating cobwebs thick as shrouds. A chilling draft whispered through the halls, carrying the scent of damp earth and something else... something metallic.

His EMF reader sputtered, the needle wildly fluctuating. Footsteps echoed from the floor above, though the house stood eerily still. A low moan, like wind through broken glass, pierced the silence. He pressed on, his heart hammering against his ribs. A shadowed figure flickered at the edge of his vision, then vanished.

He found a dusty diary detailing the manor's dark history – generations of madness and gruesome deaths. A chilling entry described a ritual sacrifice, a pact with an ancient entity. Suddenly, the temperature plummeted. He felt a presence, cold and malevolent, wrapping around him like icy tendrils.

He spun around, his breath catching in his throat. Standing before him, bathed in an unnatural light, was a figure… himself. But this Dhanush was gaunt, eyes hollow, a sinister smile twisting his lips. The doppelganger raised a hand, revealing a gleaming, wickedly sharp scalpel.

Terror choked him. This wasn't a ghost; it was a reflection of his own deepest fears, a manifestation of his obsession with the macabre. The diary fell from his trembling hands, its final entry now clear: "The house doesn't haunt you; you haunt the house." His reflection lunged, the scalpel glinting. The last thing Dhanush saw was his own terrified eyes mirroring his own impending doom.



**Conclusion:**

HauntScript showcases the effective use of AI in creative domains by streamlining and enhancing the process of horror story writing. It reduces the effort and time required to develop engaging narratives, making content creation more efficient and accessible for users. By combining user inputs with intelligent prompt generation, it delivers tailored stories that align with specific themes and tones. This not only supports writers facing creative blocks but also inspires new ideas and plotlines. HauntScript proves that AI can be a powerful collaborator in artistic expression, opening new possibilities in storytelling, filmmaking, game design, and other creative industries.

**Future Scope:**

1. Integrate voice narration to bring generated horror stories to life through audio.
2. Enable image generation that visually represents key scenes from the story.
3. Expand beyond horror by allowing users to customize genres such as thriller, mystery, or fantasy.
4. Introduce collaborative story building, where multiple users can contribute and build a story together in real-time.

**References:**

- Google cloud API  
- Streamlit Documentation  
- Python Docs