HAUNT SCRIPT

Chembian RK-22BAI1431
Sarveesh Kumar-22BAI1420
Vishwa-22BAI1466

Define Problem Statement

Objective: To define a concrete, real-world problem that combines creativity, user interaction, and artificial intelligence, resulting in a deployable and engaging application.

Detailed Explanation: The modern era of AI has opened up vast possibilities for personalization and creativity. However, much of the focus has been on analytical domains like prediction or classification. There is still untapped potential in the domain of *entertainment technology*—particularly AI-generated narratives. The goal of HauntScript is to address this gap by building a platform that allows users to experience personalized horror stories, instantly generated by a language model, based on their chosen characters and scenarios.

The problem is multifaceted:

- Users want engaging short-form content without spending hours reading.
- Content creators are exploring AI as a creative tool.
- There is a technical challenge in combining natural language generation, prompt engineering, and user interaction within a single seamless app.

We translated these ideas into a focused problem statement: "**How can we use AI to generate personalized, immersive horror stories in real-time through a user-friendly interface?**"

Project Proposal (Proposed Solution)

Goal: Develop an end-to-end pipeline that accepts basic user input (name, horror situation, number of lines) and uses a pre-trained language model to generate and display a coherent, chilling short story in real-time.

Solution Architecture:

1. **Frontend (UI)**: Use Streamlit to capture user inputs through a web-based interface with interactive widgets.
2. **Logic Layer**: Construct prompts dynamically and send them to a pre-trained AI model.
3. **Backend (AI Engine)**: Use Google's Gemini 1.5 Flash model via the Google Generative AI SDK.
4. **Display & Feedback**: Render the generated story on the frontend and optionally collect user feedback to improve future outputs.

Innovation Points:

- Minimal input for maximum output.
- High-speed response via Gemini Flash.
- Horror genre tone infused through carefully designed prompts.
- Deployable locally or on the cloud.

Initial Project Planning Report

1. Roles & Responsibilities:

- **Frontend Developer**: Designs UI layout, handles Streamlit input/output, manages widget states.
- **Backend Developer**: Constructs prompts, interfaces with Gemini SDK, handles exceptions and errors.
- **ML Integration Lead**: Selects LLM model, manages key configuration, tunes model generation parameters.
- **QA & Documentation**: Conducts functional testing, prepares planning reports, and ensures code documentation.

2. Technology Stack:

- Programming Language: **Python 3.10**
- Frameworks: **Streamlit**, **Google Generative AI SDK**
- Development Tools: **Anaconda**, **Visual Studio Code**
- Deployment Options: Localhost (via Streamlit), Cloud (optional with Streamlit Cloud or Render)
- Source Control: **GitHub repository** for collaboration and versioning

3. Timeline and Gantt-Based Milestone Chart:

- **Week 1**: Ideation and research, writing assignments, defining scope.
- **Week 2**: UI wireframing and first working version with mock outputs.
- **Week 3**: Full integration with Gemini API and input validation.
- **Week 4**: Testing and refinement of prompts, UI polishing.
- **Week 5**: Documentation, demo recording, final deployment package.

4. Risk Analysis & Mitigation:

- **Latency Risks**: Ensured by choosing Gemini Flash which is optimized for speed.
- **Token Limit Exceedance**: Prompt structure is designed to stay under 2048 tokens.
- **Dependency Conflicts**: Isolated environment using Anaconda virtual env (`horror_gen`).
- **UI Bugs**: Streamlit's real-time updates ensure easy debugging and rollback.

5. Deliverables of This Phase:

- Finalized Problem Statement (PDF)

- Project Proposal Document (PDF/MD)
- Initial Planning Report (including team roles, timeline, tools)
- GitHub repository setup and folder scaffolding

Visual Planning

- Flowchart: Input → Prompt → Gemini API → Response → Display
- UML Use Case Diagram: Actor (User), System (HauntScript), Actions (Input, Generate, Read)
- GitHub Project Board: Created with columns for "To Do", "In Progress", "Testing", "Completed"

Conclusion: The planning phase of HauntScript was crucial in ensuring a structured and organized approach to development. With clearly defined roles, an achievable timeline, and an innovative concept, we set ourselves up for success. The combination of generative storytelling with real-time delivery places this project at the intersection of entertainment and technology.