Project Report: Dungeon GPT Lite

Introduction

This report details the development of **Dungeon GPT Lite**, an interactive AI-powered story generator. My primary objective was to create a robust and easily deployable web application leveraging the Google Gemini API for dynamic storytelling. This project demonstrates a full-stack approach using Streamlit, Google Cloud Firestore, and the Gemini API, designed for rapid prototyping and seamless deployment.

Abstract

Dungeon GPT Lite is a web application facilitating collaborative AI storytelling. It addresses challenges like lack of persistent memory, limited content control, and complex deployments. By using Streamlit for a unified Python codebase, Google Cloud Firestore for data persistence, and the Google Gemini API for AI generation, the project delivers an intuitive UI with dual AI response modes (Censored/Uncensored) and comprehensive session management (save, load, import, export). Its architecture prioritizes accessibility and efficient cloud deployment, showcasing effective integration of modern AI and web technologies.

Tools Used

My development relied on a focused tech stack:

- **Core Application Framework:** Streamlit (Python) for rapid, unified frontend/backend development.
- **Artificial Intelligence:** Google Gemini API (`gemini-1.5-flash-latest`) for all text generation, with configurable safety settings.
- **Database:** Google Cloud Firestore for persistent story and session data storage.
- **Version Control:** Git and GitHub for source code management.
- **Deployment Platform:** Streamlit Cloud for streamlined, one-click public accessibility.
- **Development Environment:** Python virtual environments (`venv`) for dependency isolation.

Steps Involved in Building and Deploying the Project

My project development and deployment followed an agile methodology, focusing on iterative feature implementation and problem-solving.

Development Phase

1. **Unified Architecture & UI Scaffolding:** I began by strategically choosing Streamlit to unify frontend and backend logic in Python, enabling rapid UI development. This involved designing the interactive chat interface, input mechanisms, and control buttons for an intuitive user experience.
2. **Advanced Google Gemini API Integration:** Integrated the Gemini API using `google-generativeai`. I specifically selected `gemini-1.5-flash-latest` after testing, implementing logic to dynamically adjust Gemini's **safety settings** (Censored/Uncensored modes) and injecting user-defined "Temperature" and "Story Tone/Genre" into prompts for tailored narrative outcomes.
3. **Robust Data Persistence with Google Cloud Firestore:** Developed a robust data persistence layer using Firestore via `firebase-admin`. A key architectural decision was to associate each user's session with a unique UUID in Firestore, enabling seamless **saving and loading of entire story histories** across different sessions and devices.

4. **Comprehensive Session Management (Import/Export):** Implemented advanced session management: **export** allows downloading full chat history and settings as JSON for portability and backup, while **import** enables loading previous sessions or shared narratives from these files, enhancing versatility and user control.

Deployment Phase

1. **GitHub Repository Preparation:** Structured the project on GitHub with `streamlit_app.py`, `requirements.txt`, and a `.gitignore` to prevent sensitive file exposure, preparing for cloud deployment.
2. **Google Cloud Infrastructure Setup:** Established a new Google Cloud Project, enabled the Generative Language API, generated a secure API key, and configured Firebase (Firestore and Anonymous Authentication) to support backend services.
3. **Streamlit Cloud Configuration & Secrets Management:** Linked the GitHub repository to Streamlit Cloud. Critically, `GOOGLE_API_KEY` and the `firebase_service_account_key` were securely configured as secrets within Streamlit Cloud's "Advanced settings," ensuring proper formatting for the multi-line JSON key.
4. **Deployment Execution:** Initiated one-click deployment on Streamlit Cloud, which handled the entire build process (cloning, installing dependencies, launching the app), making Dungeon GPT Lite publicly accessible.

Challenges Encountered and Technical Solutions

Throughout development and deployment, I faced several significant technical and functional challenges, which I resolved through in-depth logical debugging and robust engineering solutions.

1. **Dynamic AI Parameter Integration & Context Management:** Effectively injecting dynamic parameters (Temperature, Tone/Genre) into the prompt while maintaining conversational coherence was complex.
    - **Technical Solution:** I engineered a robust prompt construction system, dynamically prepending persona/tone instructions. A crucial sliding window approach (last 10 messages) managed token limits. `temperature` was passed directly to Gemini's `generation_config`, enabling real-time AI creativity control for adaptable, high-quality narratives.
2. **Resolving Google Gemini API Accessibility (`404 models/gemini-pro` error):** Initial attempts to use `gemini-pro` consistently failed with a `404 Not Found` error, indicating a deeper API endpoint availability or permissioning issue.
    - **Technical Solution:** Through extensive logical debugging, I identified an environmental constraint and strategically switched to `gemini-1.5-flash-latest`. This resolved API access, demonstrating adaptability in integrating with external AI services.
3. **Preventing Infinite Loops and Ensuring Idempotency in `st.file_uploader` (Import Functionality):** The `st.file_uploader` initially caused an infinite processing loop after file upload due to its state not reliably resetting.
    - **Technical Solution:** I devised a robust mechanism by encapsulating `st.file_uploader` within an `st.form` with `clear_on_submit=True`. By linking processing to a dedicated "Process Import" button, the uploader's state was reliably cleared, guaranteeing single processing per user action and enhancing stability.
4. **Maintaining Seamless UI State Across Reruns:** Streamlit's stateless nature presented a challenge in preserving consistent UI and conversational state during script reruns.
    - **Technical Solution:** I architected the application to extensively utilize `st.session_state` to persist critical variables like `story_history`, `current_mode`, `temperature`, and `tone`. This ensured the application's visual display and underlying logic consistently reflected the true state, providing a fluid and continuous user experience.

Conclusion

Dungeon GPT Lite stands as a robust demonstration of developing a full-stack, AI-powered web application, emphasizing practical problem-solving and technical resilience. Through strategic integration of Streamlit, the Google Gemini API, and Google Cloud Firestore, I successfully built a platform delivering dynamic, interactive storytelling, addressing critical aspects like data persistence, AI control, and efficient session management. The advanced challenges encountered during API integration, Streamlit widget state management, and maintaining UI consistency provided invaluable hands-on experience in architecting and debugging complex technical issues within a modern deployment environment. This project solidifies my expertise in agile web development, generative AI, and scalable cloud database technologies.