**Design Phase 1: Gemini Landmark Explorer**

Project Phase: Design & Development (Main Folder 03)

Sub Folder: Phase 1 Design Document

Date Prepared: June 20, 2025

Project Title: Gemini Landmark Explorer – An AI-Powered Multimodal Landmark Description App

Team:

Kamlesh Chowdhary ( [kamlesh.23bce10260@vitbhopal.ac.in](mailto:kamlesh.23bce10260@vitbhopal.ac.in) )

Prashasti Joshi ( [prashasti.23bce10893@vitbhopal.ac.in](mailto:prashasti.23bce10893@vitbhopal.ac.in) )

Somil Asati ( [somil.23bce10364@vitbhopal.ac.in](mailto:somil.23bce10364@vitbhopal.ac.in) )

Devansh Tyagi ( [devansh.23bce10247@vitbhopal.ac.in](mailto:devansh.23bce10247@vitbhopal.ac.in) )

Version: 1.0

**1. Introduction**

This document outlines the initial design specifications for the Gemini Landmark Explorer application. It details the high-level system architecture, key components, data flow, user interface considerations, and the technical stack, serving as a blueprint for the development phase. The design prioritizes modularity, scalability (for the AI component), user experience, and data privacy.

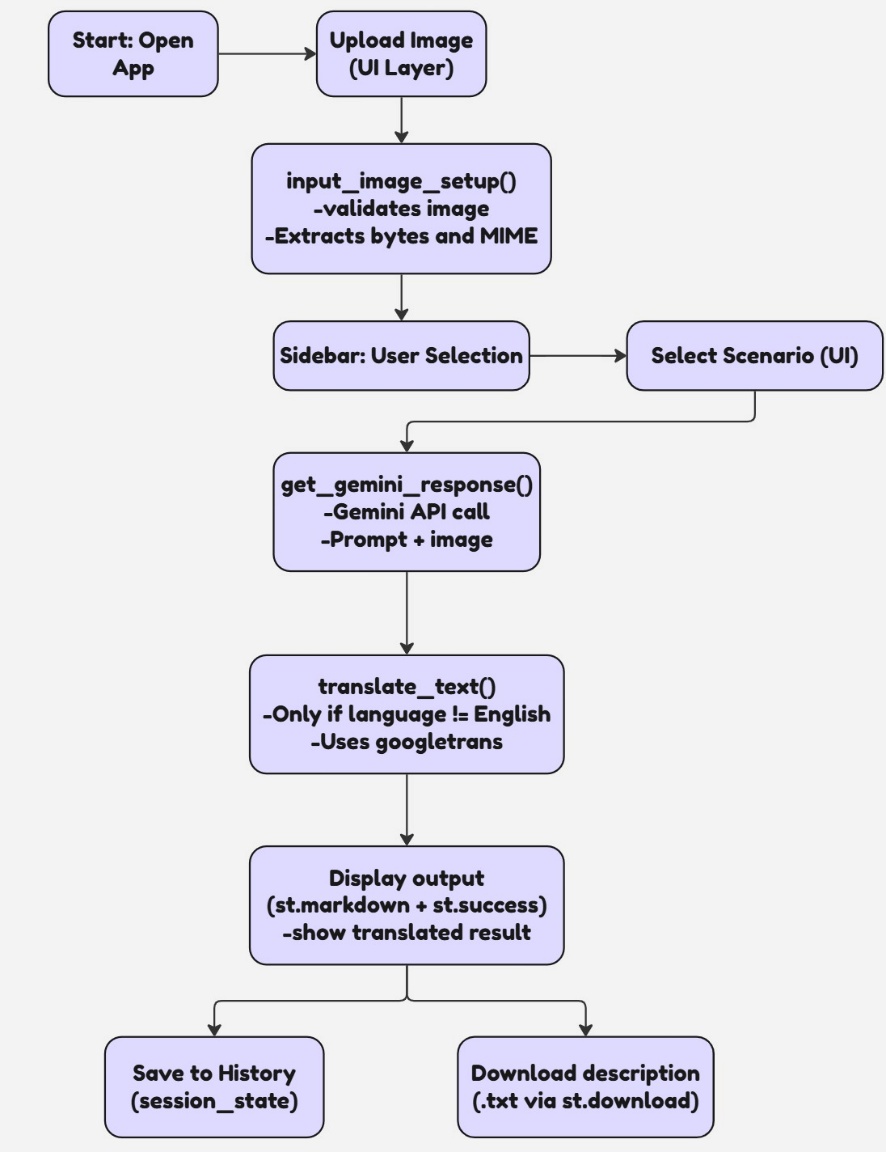
**2. System Architecture**

The Gemini Landmark Explorer follows a client-server architecture model, where the Streamlit web application acts as the client-side interface and the core logic, interacting with external AI and translation services.

High-Level Overview:

[Source: Project Report Landmark Explorer.docx, Section 5.2.1 "System Architecture: Workflow Diagram"]

* **User Interface (Client-Side):** The primary interaction point for users, built using Streamlit. Handles image uploads, text input, scenario selection, language preference, and displays AI-generated descriptions.
* **Application Logic (Server-Side - Streamlit Backend):** Processes user inputs, prepares data for API calls, orchestrates communication with external services, and manages application state (e.g., session history). This is embodied within the app.py script.
* **Google Gemini AI Model:** An external cloud-based service (specifically gemini-2.0-flash-001 or gemini-pro-vision) responsible for multimodal analysis of images and text prompts to generate landmark descriptions. Accessed via the google-generativeai Python library.
* **Google Translator Service:** An external service (accessed via the googletrans library) responsible for translating the AI-generated descriptions into the user's selected language.
* **System Workflow Diagram (or Activity Diagram):** As seen in your Project Report Landmark Explorer.docx (Section 5.2.1), this diagram is crucial. It visually depicts the sequence of operations from user input to final output:

****

**3. Component Design**

**3.1. User Interface (Streamlit Frontend)**

* **Components:**
* **Header:** st.title and st.markdown for app branding and overview [Source: Gemini\_Landmark\_Description\_App\_Final\_Documentation.docx].
* **Image Uploader:** st.file\_uploader for user image input (PNG, JPEG) [Source: app.py].
* **Optional Text Input:** st.text\_input for users to provide additional context or questions.
* **Scenario Selector:** st.sidebar.selectbox to choose predefined interaction scenarios (Traveler, Tour Guide, etc.) [Source: app.py].
* **Language Selector:** st.sidebar.selectbox for target translation language [Source: app.py].
* **Submit Button:** st.button to trigger the analysis process.
* **Output Display:** st.markdown to render the AI-generated and translated description.
* **Download Button:** st.download\_button to allow users to save the description [Source: app.py].
* **Session History:** Display of recent interactions using st.session\_state.history.
* **Feedback:** st.spinner for loading, st.success for completion, error messages.

**3.2. Application Logic (Python Backend - app.py)**

* **main() function:** Orchestrates the Streamlit UI elements and calls to other functions.
* **load\_dotenv():** Manages environment variables for API key security [Source: app.py].
* **get\_gemini\_response(image, prompt):** Core function for interacting with the Gemini API.
* Initializes genai.GenerativeModel('gemini-2.0-flash-001') [Source: app.py].
* Sends processed image data and the dynamic prompt to the model.
* Returns the AI's textual response.
* **input\_image\_setup(uploaded\_file):** Processes the uploaded image file.
* Reads bytes data from uploaded\_file.getvalue() [Source: app.py].
* Determines mime\_type and returns structured image data for the Gemini API.
* **translate\_text(text, target\_lang):** Handles text translation.
* Uses googletrans.Translator() [Source: app.py].
* Translates text if target\_lang is not English.
* **get\_image\_base64(img):** Utility for displaying images in Streamlit.
* **st.session\_state.history:** In-memory list to store recent descriptions [Source: app.py].
* **scenario\_prompts:** Dictionary holding specific prompts tailored to each user scenario [Source: app.py].

**3.3. External Services**

* **Google Gemini API:** Provides the multimodal AI capabilities.
* **Google Translate (via googletrans):** Provides text translation services.

**4. Data Flow**

1. **User Uploads Image:** An image file (e.g., JPEG, PNG) is uploaded via st.file\_uploader.
2. **Image Processing:** input\_image\_setup converts the uploaded\_file into a byte stream and identifies its MIME type, packaging it for the Gemini API.
3. **Prompt Generation:** Based on the user's selected scenario, a corresponding prompt is retrieved from scenario\_prompts. If the user provides additional text, it can be appended or incorporated.
4. **AI Request:** The processed image data and the generated prompt are sent as input to the get\_gemini\_response function, which makes an API call to the Google Gemini model.
5. **AI Response:** The Gemini model analyzes the image and prompt, generating a detailed text description of the landmark. This response is returned to the app.py script.
6. **Translation (Optional):** If a language other than English is selected, the AI-generated description is passed to the translate\_text function, which utilizes the googletrans library to translate the text.
7. **Display & Storage:** The final (translated) description is displayed on the Streamlit UI, and it's appended to st.session\_state.history for in-session access.
8. **Download:** The user can opt to download the displayed description as a .txt file.

**5. User Interface (UI) Design Principles**

The UI design focuses on simplicity, clarity, and ease of use, leveraging Streamlit's capabilities:

* **Clean Layout:** Minimalistic design to keep the focus on image upload and description output.
* **Intuitive Workflow:** A clear step-by-step process: Upload -> Select Options -> Discover.
* **Contextual Guidance:** Improved placeholder texts and header messages guide the user effectively [Source: Gemini\_Landmark\_Description\_App\_Final\_Documentation.docx].
* **Responsiveness:** Streamlit inherently provides a degree of responsiveness for different screen sizes.
* **Feedback:** Visual cues like spinners and success messages keep the user informed during processing.

**6. Technical Stack**

* **Programming Language:** Python 3.9+
* **Web Framework:** Streamlit
* **AI Model:** Google Gemini Pro Vision (gemini-2.0-flash-001)
* **AI Library:** google-generativeai
* **Image Processing:** PIL (Pillow)
* **Translation:** googletrans
* **Environment Management:** python-dotenv
* **Version Control:** Git (e.g., GitHub)

**7. Design Considerations & Diagrams**

* **Modularity:** The design separates concerns into distinct functions (image setup, API calls, translation) for easier maintenance and testing.
* **Security:** API keys are managed via environment variables and images are processed in-memory, ensuring sensitive information is not exposed or persistently stored.
* **Scalability:** While the current implementation is a single-file Streamlit app, the reliance on cloud APIs (Gemini) means the core AI processing is inherently scalable.
* **Diagrams for Design:**
* **System Workflow Diagram (as mentioned in Section 2):** Essential for understanding the overall process flow.
* **Component Diagram (High-Level):** Illustrates the main software components (Streamlit App, Gemini API, Google Translator) and their relationships.
* **Sequence Diagram (Optional for Phase 1):** Could detail the exact order of messages and interactions between the Streamlit app and the Gemini/Translator APIs for specific operations.