**Design Phase 2: Gemini Landmark Explorer**

Project Phase: Design & Development (Main Folder 03)

Sub Folder: Phase 2 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 elaborates on the detailed design aspects of the Gemini Landmark Explorer application, building upon the high-level architecture defined in Design Phase 1. It covers the specific design of key modules and functions, error handling, security measures, performance considerations, and testing strategies. The aim is to provide a granular blueprint for implementation, ensuring robustness, usability, and adherence to project goals.

**2. Detailed Module and Function Design**

The core logic of the application is encapsulated within app.py, leveraging modular functions for clarity and maintainability.

**2.1. get\_gemini\_response(image, prompt)**

* **Purpose:** Interfaces directly with the Google Gemini AI model to obtain landmark descriptions.
* **Inputs:**
* image: A list containing a dictionary with mime\_type (e.g., "image/jpeg") and data (bytes of the image).
* prompt: A string containing the textual instruction for the Gemini model.
* **Processing:**
* Instantiates the generative model: model = genai.GenerativeModel('gemini-2.0-flash-001') [Source: app.py]. The model choice is gemini-2.0-flash-001 for its speed and multimodal capabilities, suitable for quick responses.
* Calls model.generate\_content([prompt, image[0]]) to send both text and image to the AI.
* Extracts the text content from the model's response.
* **Outputs:** A string containing the AI-generated description of the landmark.
* **Error Handling (Internal):** Assumes the Gemini API call is successful; external error handling is done at the calling UI layer.

**2.2. input\_image\_setup(uploaded\_file)**

* **Purpose:** Prepares the uploaded image file into a format suitable for the Gemini API.
* **Inputs:** uploaded\_file: A Streamlit UploadedFile object.
* **Processing:**
* Checks if uploaded\_file is not None.
* Retrieves raw bytes data using uploaded\_file.getvalue().
* Extracts the MIME type from uploaded\_file.type.
* **Outputs:** A list containing a dictionary with mime\_type and data keys, ready for the Gemini model.
* **Error Handling:** Raises FileNotFoundError("No file uploaded") if uploaded\_file is None [Source: app.py]. This is caught at the UI level to display a user-friendly message.

**2.3. translate\_text(text, target\_lang)**

* **Purpose:** Translates the AI-generated description into the user's chosen language.
* **Inputs:**
* text: The string to be translated (AI-generated description).
* target\_lang: The two-letter language code (e.g., "en", "hi", "es") for the target language.
* **Processing:**
* Checks if target\_lang is "en" (English); if so, returns the original text without translation to avoid unnecessary API calls [Source: app.py].
* Initializes googletrans.Translator() (note: googletrans uses an unofficial Google Translate API, which can have rate limits or unreliability).
* Calls translator.translate(text, dest=target\_lang).text.
* **Outputs:** Translated text string.
* **Error Handling:** Relies on googletrans internal error handling; robust error handling for googletrans is a known challenge due to its unofficial nature.

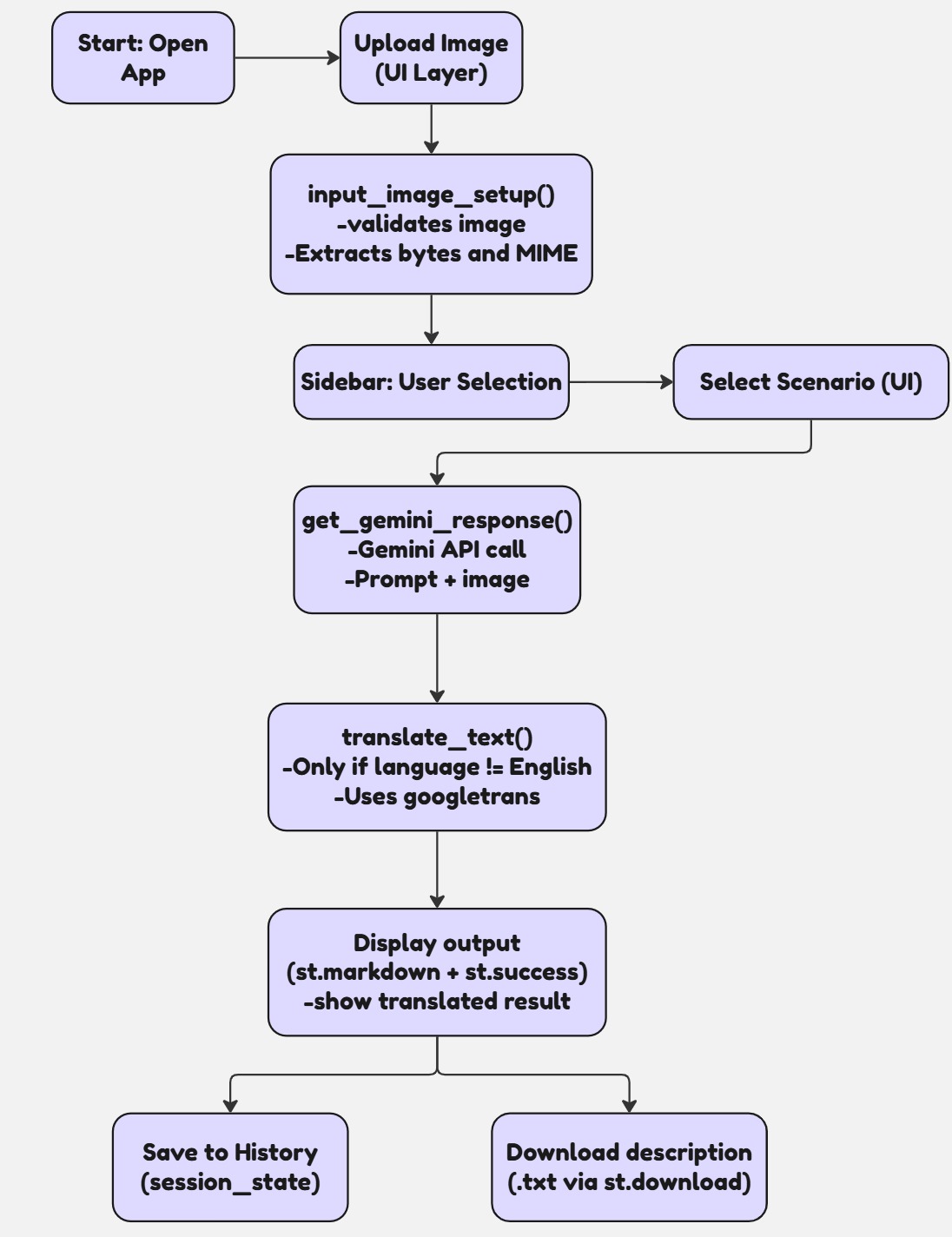
**2.4. get\_image\_base64(img)**

* **Purpose:** Utility function to convert a PIL Image object to a base64 string for display in Streamlit via unsafe\_allow\_html=True [Source: app.py].
* **Inputs:** img: A PIL (Pillow) Image object.
* **Processing:** Saves the image to a BytesIO buffer in PNG format, then encodes it to base64.
* **Outputs:** Base64 encoded string of the image.

**2.5. Streamlit UI Logic (app.py main block)**

* **Session State:** st.session\_state.history is initialized to an empty list to store descriptions within the current user session [Source: app.py].
* **Scenario Prompts:** scenario\_prompts dictionary defines specific initial prompts for the Gemini model based on the selected user scenario [Source: app.py].
* **User Interaction Flow:**
* Image upload handled by st.file\_uploader.
* Optional text input by st.text\_input.
* Scenario and language selection via st.sidebar.selectbox.
* "Discover Landmark Info" st.button triggers the core logic:
* Calls input\_image\_setup.
* Retrieves dynamic prompt from scenario\_prompts.
* Displays st.spinner during AI processing.
* Calls get\_gemini\_response.
* Calls translate\_text.
* Displays st.success and the result via st.markdown.
* Appends result to st.session\_state.history.
* Provides st.download\_button for saving the output.

**2.6 System Flowchart**

****

**3. Error Handling Design**

* **User Input Validation:** Basic validation for image upload (checks if uploaded\_file:).
* **Exception Handling:**
* try-except block around the AI processing (get\_gemini\_response) and translation calls to catch potential errors (e.g., FileNotFoundError if no image, or API-related errors).
* User-friendly error messages are displayed using st.error in case of issues (e.g., "Please upload an image first.").
* **Feedback:** Visual spinners (st.spinner) and success messages (st.success) are used to inform the user about the application's state, preventing perceived unresponsiveness.

**4. Security Design**

* **API Key Management:**
* The Google API Key is stored in a .env file and loaded using python-dotenv [Source: app.py].
* os.getenv("GOOGLE\_API\_KEY") is used to retrieve the key, ensuring it's not hardcoded in the source.
* The .env file should be excluded from version control (e.g., via .gitignore).
* **Data Privacy:**
* Crucially, user-uploaded images are processed *in memory* [Source: Gemini\_Landmark\_Description\_App\_App\_Final\_Documentation.docx, Project Report Landmark Explorer.docx].
* No user images or generated descriptions are persistently stored on the application's server or database. This minimizes privacy risks.
* **Input Sanitization:** While not explicitly detailed, standard practices of input validation should be implicitly considered, especially if user prompts were to be used in ways beyond direct API calls to prevent injection vulnerabilities (though less critical for this specific app structure).

**5. Performance Considerations**

* **Response Time Goal:** The project aims for a response time under 10 seconds for typical inputs [Source: Project Report Landmark Explorer.docx].
* **API Latency:** Performance is heavily dependent on the response time of the external Google Gemini API and the Google Translate service.
* **Image Size:** Larger image files will naturally take longer to upload and process. The design implicitly handles this by processing bytes directly.
* **Streamlit Performance:** Streamlit handles caching for certain elements, but complex re-renders on every interaction could impact perceived speed. The current design limits intensive computation to explicit button clicks.
* **Model Choice:** The use of gemini-2.0-flash-001 suggests a preference for speed over maximum complexity compared to larger models.

**6. Testing Strategy**

* **Unit Testing (Conceptual):**
* **get\_gemini\_response:** Test with mock API responses to ensure correct parsing and return format. Test with various prompt and image data combinations.
* **input\_image\_setup:** Test with valid and invalid (e.g., None) uploaded files to ensure correct data preparation and error handling.
* **translate\_text:** Test with various texts and target languages, including English (which should bypass translation). Test edge cases like empty strings.
* **Integration Testing:**
* Verify the end-to-end flow from image upload to displaying the AI-generated and translated text.
* Test integration with actual Gemini API.
* Test integration with googletrans.
* **User Acceptance Testing (UAT):**
* Verify scenario-based prompt customization (e.g., does "Tour Guide" give different output than "Educator"?).
* Test multilingual functionality with native speakers if possible.
* Check UI responsiveness and user experience on different devices.
* Verify download functionality and session history.
* **Performance Testing:** Measure actual response times against the 10-second goal using various image sizes and network conditions.

**7. Future Enhancements (Brief)**

While out of scope for this phase, detailed design for future versions could include:

* Integration with official Google Cloud Translation API for more reliable and high-volume translation.
* Database integration for persistent storage of user preferences, generated history (opt-in), or a library of landmark data.
* More advanced input validation or image pre-processing.
* Voice input/output features.