

GEMINI HISTORICAL ARTIFACT DESCRIPTION

Team ID : LTVIP2026TMIDS77577

TEAM MEMBERS:

Kudavalli Sujitha

Lakshmi Durga Bhavani Sankar Gunturu

Borra Vikash

Lakshman Seshapu

PROBLEM STATEMENT:

Historical artifacts hold great cultural and educational value, but they often lack clear descriptions especially in digital platforms or museums. This project addresses that problem by providing accurate and precise description of the artifact from image and avoiding the need of manually maintaining details about the artifact. This helps the educators, students, and curators better understand and document the artifacts from all around the world.

REQUIREMENTS:

● Functional Requirements:

The interface should allow users do the following tasks:

- Uploading the image of an artifact (.png, .jpeg, .jpg).
- Input the image along with optional prompts by the user.
- Generate a description of the artifact given by the user.

- Download the description generated as a text file.

- **Non-Functional Requirements:**

The non-functional requirements include:

- Providing an accurate description to the user prompt.
- User should easily be able to access the features of the artifact description through a user-friendly interface along with safety integrated through API keys and environment variables.

PROJECT OVERVIEW:

The web application is built using Streamlit which allows the users to upload an image of the historical artifact to receive an AI generated description. It uses Gemini 1.5 Flash model from Google's Generative AI API's; it supports both texts and images.

PROJECT STRUCTURE:

The project folder includes the following files:

- **requirements.txt**

The file contains the Python libraries and packages needed to be installed for the project

➤ **README**

Provides an overview of the project.

➤ **app.py**

This Python folder contains all the main code which helps make this generative model work

➤ **.env**

Helps in storing sensitive and vulnerable information such as API keys.

CODE:

● Requirements.txt

```
Streamlit
google-generativeai
Pillow
python-dotenv
```

● app.py

```
from dotenv import load_dotenv
import streamlit as st
import os
import google.generativeai as genai
from PIL import Image

# --- Configuration ---
load_dotenv()
api_key = os.getenv("GEMINI_API_KEY")
genai.configure(api_key=api_key)

# --- Utility Functions ---
def input_image_setup(uploaded_file):
    if uploaded_file is not None:
        bytes_data = uploaded_file.getvalue()
```

```
        return [{"mime_type": uploaded_file.type, "data": bytes_data}]
    else:
        raise FileNotFoundError("No file uploaded")

def get_gemini_response(input_text, image_data, prompt):
    full_prompt = f"{prompt}\n\nUser Instruction: {input_text}"
    model = genai.GenerativeModel("gemini-1.5-flash")
    response = model.generate_content([full_prompt, image_data[0]])
    return response.text

# --- Streamlit Page Settings ---
st.set_page_config(
    page_title="💡 Gemini historical Artifact Description
App", page_icon="💡",
    layout="centered",
)

# --- Custom CSS Styling ---
st.markdown("""
<style>
    .stApp {
        background-color: #ecf0f3;
        font-family: 'Verdana', sans-serif;
        color: #222222;
    }

    .title {
        font-family: 'Georgia', serif;
        font-size: 36px;
        font-weight: 700;
        color: black;
        text-align: center;
        margin-top: 0.5rem;
        margin-bottom: 0.2rem;
    }

    section[data-testid="stSidebar"] {
        background: linear-gradient(to bottom, #2c3e50, #34495e);
        color: #ffffff;
    }

    section[data-testid="stSidebar"] h1,
    section[data-testid="stSidebar"] h2,
    section[data-testid="stSidebar"] h3,
    section[data-testid="stSidebar"] h4,
    section[data-testid="stSidebar"] h5,
    section[data-testid="stSidebar"] h6,
    section[data-testid="stSidebar"] label,
    section[data-testid="stSidebar"] span {
        color: #ffffff !important;
        font-weight: bold;
    }
</style>
""")
```

```
main[data-testid="stAppViewContainer"] {  
    background-color: #ecf0f3;  
}  
  
main[data-testid="stAppViewContainer"] > div:first-child {  
    padding: 0 !important;  
    margin: 0 !important;  
}  
  
.stButton button {  
    background-color: #4B47C3;  
    color: white;  
    border-radius: 8px;  
    font-weight: bold;  
    padding: 0.5em 1.4em;  
    transition: background-color 0.3s ease;  
}  
  
.stButton button:hover {  
    background-color: #3934a1;  
}  
  
.stTextInput input {  
    background-color: #ffffff;  
    border: 1px solid #ffffff;  
    border-radius: 6px;  
    padding: 0.5em;  
    color: #000000;  
}  
  
.stTextArea textarea {  
    background-color: #ffffff;  
    border: 1px solid #ffffff;  
    border-radius: 6px;  
    padding: 0.5em;  
    color: #000000;  
}  
  
.stMarkdown {  
    background-color: white;  
    padding: 1.2rem;  
    border-radius: 12px;  
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);  
    margin-top: 1rem;  
}  
  
.stDownloadButton button {  
    background-color: #6E6E6E;  
    color: white;  
    border-radius: 6px;  
}
```

```

        .stDownloadButton button:hover {
            background-color: #4f4f4f;
        }

        .css-1d391kg, .css-1dp5vir {
            background-color: #ffffff !important;
            border-radius: 12px;
            padding: 1rem !important;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.06);
        }

        .css-qrbaxs, .css-1v3fvcr {
            color: #4B47C3 !important;
            font-weight: 600 !important;
        }
    </style>
""", unsafe_allow_html=True)

# --- Header ---
st.markdown('<div class="title"> Gemini historical Artifact Description App </div>', unsafe_allow_html=True)

# --- Sidebar Input Section ---
with st.sidebar:
    st.header("💡 Upload Inputs")
    uploaded_file = st.file_uploader("Upload Artifact Image", type=["jpg", "jpeg", "png"])
    input_text = st.text_area("💡 Add Context or Custom Prompt (optional)", placeholder="Describe the context, location, or your question...")
    submit = st.button("💡 Generate Description")

# --- Display Uploaded Image ---
if uploaded_file:
    st.image(uploaded_file, caption="💡 Uploaded Artifact", use_container_width=True)

# --- Predefined System Prompt ---
system_prompt = """
You are a historian. Please describe the historical artifact in the image and provide detailed information, including its name, origin, time period, material, cultural significance, and any other relevant facts.
"""

# --- Output Area ---
if submit:
    try:
        with st.spinner("💡 Analyzing artifact and generating insights..."):
            image_data = input_image_setup(uploaded_file)
            response = get_gemini_response(input_text, image_data, system_prompt)
    
```

```

# ✅ Soft Pastel Blue Full-Width Container Box
st.markdown("""
<div style='
    width: 100%;
    background-color: #d8f0ff;
    padding: 1rem 1.5rem;
    border-radius: 12px;
    border: 1px solid #b9e6fa;
    color: #114a63;
    font-family: Verdana, sans-serif;
    font-weight: 500;
    text-align: center;
    box-sizing: border-box;
    margin: 1.5rem auto;
    box-shadow: 0 2px 8px rgba(0,0,0,0.05);
'>
    ✅ Description generated successfully!
</div>
""", unsafe_allow_html=True)

st.markdown("### 🎨 AI-Generated Artifact Description")
st.markdown(response)

st.download_button(
    label="📥 Download Description",
    data=response,
    file_name="artifact_description.txt",
    mime="text/plain"
)
except Exception as e:
    st.error(f"⚠️ Error: {str(e)}")

# --- Footer ---
st.markdown("---")
st.markdown(
    "<p style='text-align: center;'>Built using Google Gemini 1.5 Flash ·
Streamlit UI .</p>",
    unsafe_allow_html=True
)

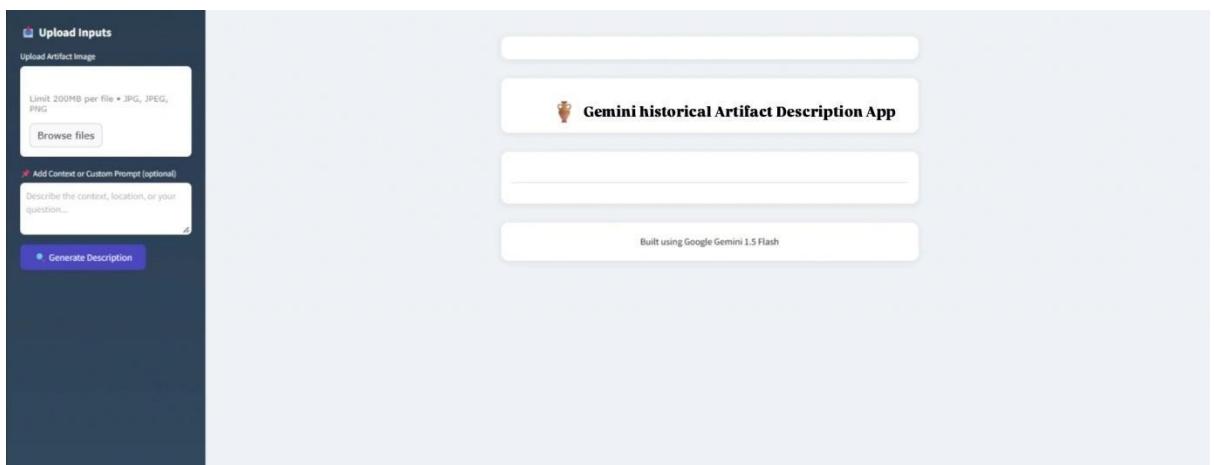
```

CODE OVERVIEW:

- The necessary libraries and packages are present in the requirements.txt

- The .env file store the API key and it securely loaded and setup for the project.
- The input_image_setup() function prepares the uploaded image so it can be sent to the API.
- The get_gemini_response() function accepts the user input , sends them to the API and returns the generated description.
- Streamlit allows the interface to accept user input along with prompt and view the description generated in the main area of the page.
- CSS is used to style the website to give it a more user-friendly interface.
- The error is displayed if the user does not upload any image while trying to generate a description.

OUTPUT PAGE:



The picture will appear in the main frame once inserted. After the generate description is pressed, the description of the artifact along with the downloadable text file is provided below the image in the main frame.

TESTING:

- Case1:When only image is uploaded

The screenshot displays the Gemini historical Artifact Description App interface. On the left, there is a dark blue sidebar with the title "Upload Inputs" and a sub-section "Upload Artifact Image". It includes a file upload input field with a limit of 200MB per file (JPG, JPEG, PNG), a "Browse files" button, and a "Screenshot 2025-06" preview. Below this is an optional text input field for "Add Context or Custom Prompt (optional)" with the placeholder "Describe the context, location, or your question...". At the bottom of the sidebar is a purple "Generate Description" button. The main right area shows a bronze ushabti figurine of an ancient Egyptian deity. Above the image is the app's logo and the text "Gemini historical Artifact Description App". To the right of the image is a light blue success message box containing the text "Description generated successfully!". Below this is a white box titled "AI-Generated Artifact Description" which contains the generated text. The text describes the artifact as a bronze ushabti from the New Kingdom, Late Period, mentioning its function as a servant in the afterlife and its small size. It also notes the lack of visible inscriptions and the material being bronze.

The description is successfully generated in Case 1.

- Case2: When image is uploaded along with prompt.

The screenshot displays two separate instances of the Gemini historical Artifact Description App interface.

Case 1 (Top): An image of a bronze sculpture of Shiva Nataraja, the cosmic dancer, is uploaded. The interface shows the uploaded file "wp11244402.jpg". A text input field contains the placeholder "Describe this sculpture, its symbolism, historical period, and significance." A purple "Generate Description" button is visible at the bottom.

Case 2 (Bottom): The same image is uploaded again. This time, the "Generate Description" button is highlighted in blue. To the right, a success message "Description generated successfully!" is displayed above the "AI-Generated Artifact Description" section. The description text is as follows:

That's a bronze sculpture depicting Shiva Nataraja, the cosmic dancer.
Name: Nataraja (meaning "King of Dance")
Origin: Likely from South India, a region with a strong tradition of bronze casting and Hindu iconography. The precise origin within South India would be difficult to determine without more information (such as maker's marks or provenance).
Time Period: While the exact date is impossible to ascertain from the image alone, the style suggests it is a modern reproduction of a classic form. Authentic Nataraja sculptures in this style were frequently made during the Chola period (approximately 9th–13th centuries CE), a time renowned for its exceptional bronze casting. Modern reproductions are quite common and are often sold as religious artifacts or decorative pieces. Therefore, determining the specific time period requires more evidence.
Material: Bronze. The rich golden patina suggests a deliberate aging process or the use of specific alloys to achieve that effect.
Cultural Significance: Nataraja is one of the most important and widely recognized representations of Shiva, a principal deity in Hinduism. The dance itself symbolizes the cyclical nature of creation and destruction, the cosmic dance of the universe. The various elements of the sculpture hold deep symbolic meaning:

- The Dance:** Represents the continuous cycle of creation and destruction, birth and death, and the ever-changing nature of existence.

The description is successfully generated in Case 2.

- **Case3 : When only prompt is given or when the image limit exceeds .**

In the above case, the interface produces an error.



ADVANTAGES:

- User-Friendly interface
- Generates the description within seconds

- Useful for people who want to educate themselves on artifacts and don't have the resources.

NON-ADVANTAGES:

- Dependency on an external API.
- AI might face the problem of hallucination without proper access to current data.

CONCLUSION:

The Gemini Historical Artifact Describer demonstrates how generative AI can support cultural heritage documentation by creating artifact descriptions from images and user prompts. This project shows the potential of AI to help educators and the public in understanding more about historical artifacts at the ease of their fingertips.