GENERATIVE AI WITH GOOGLE CLOUD

Gemini Historical Artifact Description App

1. Introduction

• Project Title: Gemini Historical Artifact Description App

• Team Members:

Name	Role
Cherukuri Lakshmi Keertana	Team Leader, Backend & AI Integration
Sandhya Medapati	Frontend UI Development
Pravallika Nekkanti	Prompt Engineering & Testing
Prajwal Dixit	API Integration & Deployment
Sonti Naga Tulasi	Documentation & QA

2. Project Overview

• Purpose

The purpose of this project is to develop an AI-powered web application that generates structured and engaging historical artifact descriptions using Google Gemini Generative AI.

The system allows users to:

- Enter artifact name
- Select word count
- Generate structured content
- Download generated blog

• Features

- AI-powered blog generation
- Custom word count selection
- Structured output (Introduction, Background, Significance, Conclusion)
- Random historical fact display
- Download as text file
- Error handling & input validation
- Cloud deployment ready

3. Architecture

• Frontend

The frontend is developed using **Streamlit**, which provides:

- Input fields (artifact name & word count)
- Generate button
- Content display area

- Download button
- Loading spinner & info messages

Frontend communicates directly with backend logic inside the Streamlit app.

• Backend

The backend is implemented using:

- Python
- Google Generative AI SDK
- Prompt engineering logic

Responsibilities:

- Accept user input
- Construct structured prompt
- Send request to Gemini API
- Receive generated content
- Return response to UI

• Database

This project does not use a persistent database (stateless architecture).

Data handling:

- Temporary in-memory processing
- Generated content stored temporarily
- Download option provided

Future scope may include MongoDB for storing user history.

4. Setup Instructions

• Prerequisites

- Python 3.9+
- pip
- Google Gemini API Key
- Virtual environment (recommended)

• Installation Steps

1. Clone the repository:

```
git clone <repository_url>
cd project-folder
```

2. Create virtual environment:

```
python -m venv venv
```

3. Activate virtual environment:

Windows:

```
venv\Scripts\activate
```

Mac/Linux:

```
source venv/bin/activate
```

4. Install dependencies:

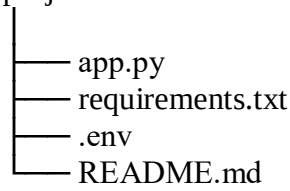
```
pip install -r requirements.txt
```

5. Create .env file:

GOOGLE_API_KEY=your_api_key_here

5. Folder Structure

project-folder/



• Client

Streamlit UI is defined inside app.py.

• Server

Backend logic and API calls are handled inside app.py.

6. Running the Application

Start the application locally:

streamlit run app.py

Application will run at:

http://localhost:8501

7. API Documentation

Base API Used:

Google Gemini Generative AI

Model:

gemini-pro

Example API Call:

response = model.generate_content(prompt)

Input:

- Prompt (string)

Output:

- Generated blog content (text)

8. Authentication

This application uses:

- Google API Key authentication
- API key stored securely in .env file
- Environment variables used for security

No user login system is implemented in current version.

9. User Interface

The UI includes:

- Artifact name input box
- Word count selector
- Generate button
- Historical fact display
- Blog output section
- Download button

10. Testing

Testing includes:

- Functional Testing (Input validation, blog generation)
- API Integration Testing
- Performance Testing
- Error Handling Testing
- Deployment Testing

All test cases passed successfully during UAT.

11. Known Issues

- Blog generation time depends on API response.
- No persistent storage for generated content.
- Requires internet connection for API access.

12. Future Enhancements

- Image-based artifact description
- Multi-language support
- User login & history tracking
- Save blogs in database
- Export to PDF/Word format
- AI content refinement options