

Team Members

Anushka Anil Rangari (ui2907)

Bindu Rajya Lakshmi Tekumudi (bi5962)

Vandana Sai Sreelekha Guntu (eb2975)

Teja Sai Nath Reddy Tatireddy (sy8143)

Course Number – CS651 Web-Based Systems

Name of Application- AI powered visual storytelling Journal

App url-<https://aipoweredjournal.uc.r.appspot.com/?view=home>

Project 2 - overview

By Group1

App Name: AI Vision Story

App URL: <https://aipoweredjournal.uc.r.appspot.com/>

App Purpose: The **AI Vision story** is a web application designed to transform personal journaling into a creative experience. It allows users to upload images—whether personal photos or images imported directly from Google Photos and uses Artificial Intelligence to automatically generate compelling narratives based on those visuals. The app serves as a digital storyteller, helping users document their memories with rich, AI-crafted stories that can be customized by tone (e.g., Formal, Poetic) and perspective. It acts as a bridge between visual memories and written expression, making journaling effortless and engaging.

Social Network Integration: The app integrates directly with Google Photos, letting users browse and import images from their cloud library without manual uploads everytime. This streamlines the experience by connecting their existing digital memories straight to the AI storytelling engine.

Project 2 - overview

Google Cloud Vision API Usage: The app leverages the **Google Cloud Vision API** to "see" and understand the user's uploaded images. When a user uploads a photo, the Vision API analyzes it to detect:

Labels: Identifying general concepts (e.g., "beach," "sunset," "party").

Objects: Locating specific items within the image (e.g., "person," "dog," "car"). This analysis provides a structured textual description of the visual content, which serves as the foundational "context" for the story generation.








Google Gemini Usage: **Google Gemini** takes the structured data from the Vision API (labels and objects) along with user-defined preferences (Tone, Perspective, Line Count, and optional Context) to craft the final narrative. Gemini weaves the disparate visual elements into a cohesive, human-like story that matches the user's desired mood and style, turning raw data into art.

Project 2 - overview

Status: Brief description of what is and is not working as it relates to your proposal.

In the proposal phase of the AI-Powered Journal, we were able to implement key features such as creating new stories, integrating a media library, importing photos from Google Photos, and saving or regenerating journal entries. However, we could not implement the planned export functionality to Instagram or Facebook due to technical and API limitations.

Are the following working:

-  Frontend (React) — **working**
-  Backend (Express/NodeJS) — **working**
-  Social Network X Authentication + User Photo Retrieval — **working**
-  Google Cloud Vision API calls — **working**
-  Google Gemini API calls — **working**
-  Google Analytics — **working**
-  Google Cloud logging — **working**

Proposal

Githublink: <https://github.com/anushkaDev9/AI-powered-visual-storytelling-telling-journal-code/wiki/Proposal>

Deployment

App url-<https://aipoweredjournal.uc.r.appspot.com/?view=home>

Run Demonstration

Now run app & functionality of app and fully demonstrate it

Google Cloud Console/Dashboard

3.Demonstration of Application

Dashboard – App Engine – AIP

AI-powered-visual-storytelling

Image to story flow

Project 2

CS651

cons^{ole}.cloud.google.com/appengine?project=aipoweredjournal&supportedpurview=project.organizationId.folder

Start your Free Trial with \$300 in credit. Don't worry—you won't be charged if you run out of credits. [Learn more](#)

Dismiss

Start free

LEARN Tutorial

Google Cloud

AIPoweredJournal

lo

Search

◆

📄

🔔

?

⋮

A

App Engine / Dashboard

Dashboard

Learn

Service

All services

Chart settings

Summary

✓ 1 hour

6 hours

12 hours

1 day

2 days

4 days

7 days

14 days

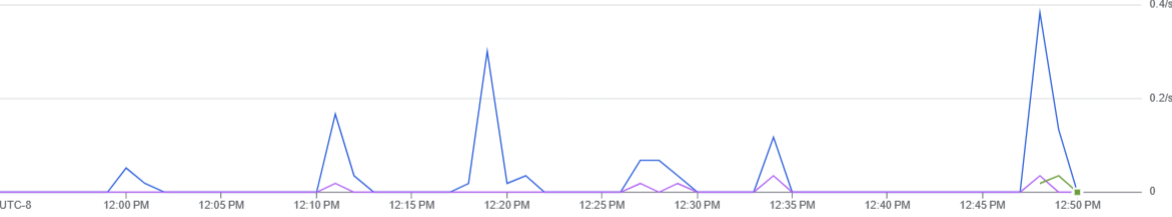
30 days

Summary

🔍

🔧

⋮



UTC-8 12:00 PM 12:05 PM 12:10 PM 12:15 PM 12:20 PM 12:25 PM 12:30 PM 12:35 PM 12:40 PM 12:45 PM 12:50 PM

Client (4XX): 0 Server (5XX): 0 Total requests: 0

Instance summary

?

Runtime support status

?

✓ All your versions are up-to-date and have no warnings.

Overview of App Engine

App Engine overview

Help document

App Engine is a fully managed, serverless platform for developing and hosting web applications at scale.

App Engine concepts

Help document

Learn about the fundamental concepts of App Engine.

App Engine Pricing

Help document

App Engine pricing.

Quotas

Help document

Lists the quotas and limits that apply to App Engine.

All App Engine documentation

Release Notes

<1

12:53

01-12-2025

Google Cloud data(base) solution

Structure of the database

the root collection is users

Each document inside users represents one authenticated user

Fields inside a users document:

Fields	Datatype
email	string
lastLoginAt	timestamp
name	string
picture	string
provider	string
updatedAt	timestamp

Under each user document you also have a subcollection called stories

Each document in stories belongs to that user only

Fields inside each story document:

Fields	Datatype
CreatedAt	timestamp
image	string(Base4)
images	array
narrative	string

Google Cloud data(base) solution

Specify Queries

- Query to create a firestore

```
export const db = new Firestore({ projectId: serviceAccount.project_id, credentials: { client_email: serviceAccount.client_email, private_key: serviceAccount.private_key, }, });
```

- Query to store users in firestore

```
export async function upsertUser(sub, data) { await db.collection("users").doc(sub).set( { ...data, provider: "google", lastLoginAt: new Date(), updatedAt: new Date(), }, { merge: true } ); }
```

- Query to check if user exists in firestore

```
export async function userExistsByEmail(email) { const snap = await db .collection("users") .where("email", "=", email) .limit(1) .get(); return !snap.empty; }
```

- Query to save the story of user in firestore

```
export async function saveStoryEntry(userId, data) { await db .collection("users") .doc(userId) .collection("stories") .add({ ...data, createdAt: new Date(), }); }
```

- Query to get user stories from firestore

```
export async function getUserStories(userId) { const snap = await db .collection("users") .doc(userId) .collection("stories") .orderBy("createdAt", "desc") .get(); return snap.docs.map((d) => ({ id: d.id, ...d.data(), createdAt: d.data().createdAt?.toDate?.() ?? null, })); }
```

- Query to delete story from firestore

```
export async function deleteStoryEntry(userId, storyId) { await db .collection("users") .doc(userId) .collection("stories") .doc(storyId) .delete(); }
```

Google Cloud Vision solution

How the vision api works

After the user uploads an image and clicks the Generate button, the Vision API analyzes the image and returns the detected objects and labels, which are then passed to the Gemini API, enabling it to generate a story based on the visual content.

Vision Api output

```
2025-12-01 11:58:44.684 === VISION OUTPUT === Image 1: Labels: Football, Ball, Football, Shorts, Ball game, Soccer ball, Player, Team sport, Soccer player, Sports. Objects: Top, Ball, Football, Person, Ball, Shorts, Person, Clothing, Top, Clothing.
```

Explain this log entry Copy Expand nested fields Hide log summary

```
{
  insertId: "692df374000a7133a61eecfb"
  labels: {1}
  logName: "projects/aipoweredjournal/logs/stdout"
  receiveTimestamp: "2025-12-01T19:58:44.831432893Z"
  resource: {2}
  textPayload:
    "=== VISION OUTPUT === Image 1: Labels: Football, Ball, Football, Shorts, Ball game, Soccer ball, Player, Team sport, Soccer player, Sports. Objects: Top, Ball, Football, Person, Ball, Shorts, Person, Clothing, Top, Clothing."
  timestamp: "2025-12-01T19:58:44.684339Z"
}
```

Google Gemini solution

How the gemini api works

After the user provides inputs such as tone, perspective, and an optional prompt, the Vision API analyzes the uploaded image and returns the detected objects and labels; these, combined with the user's inputs, are then sent to the Gemini API, which generates and returns the final story.

Vision Api output

> * 2025-12-01 11:58:44.684

=== PROMPT SENT TO GEMINI ===

> * 2025-12-01 11:58:44.684

Write a story in a **formal** tone and **first** person perspective.

> * 2025-12-01 11:58:44.684

The story must be exactly **5 lines**.

> * 2025-12-01 11:58:44.684

Here is the description of the images provided:

> * 2025-12-01 11:58:44.684

Image 1: Labels: Football, Ball, Football, Shorts, Ball game, Soccer ball, Player, Team sport, Soccer player, Sports. Objects: Top, Ball, Football, Person, Ball, Shor_

> * 2025-12-01 11:58:44.684

Rules:

> * 2025-12-01 11:58:44.684

- Exactly 5 lines.

> * 2025-12-01 11:58:44.684

- Each line must be a complete sentence.

> * 2025-12-01 11:58:44.684

- Weave the elements from all images together into a cohesive story.

> * 2025-12-01 11:58:44.684

✓ * 2025-12-01 11:58:58.271

=== STORY GENERATED === I recall that moment of profound concentration upon the verdant training pitch. Clad in my team's official orange top and white shorts, my singular objective was to master the difficult volley. With calculated effort, I raised my leg to meet the perfectly weighted pass from my colleague. The football connected precisely with my boot, ascending gracefully against the vast azure sky. In that suspended instant, I felt the pure satisfaction of a flawlessly executed maneuver.

📖 Explain this log entry

📄 Copy

⌵ Expand nested fields

🗑 Hide log summary

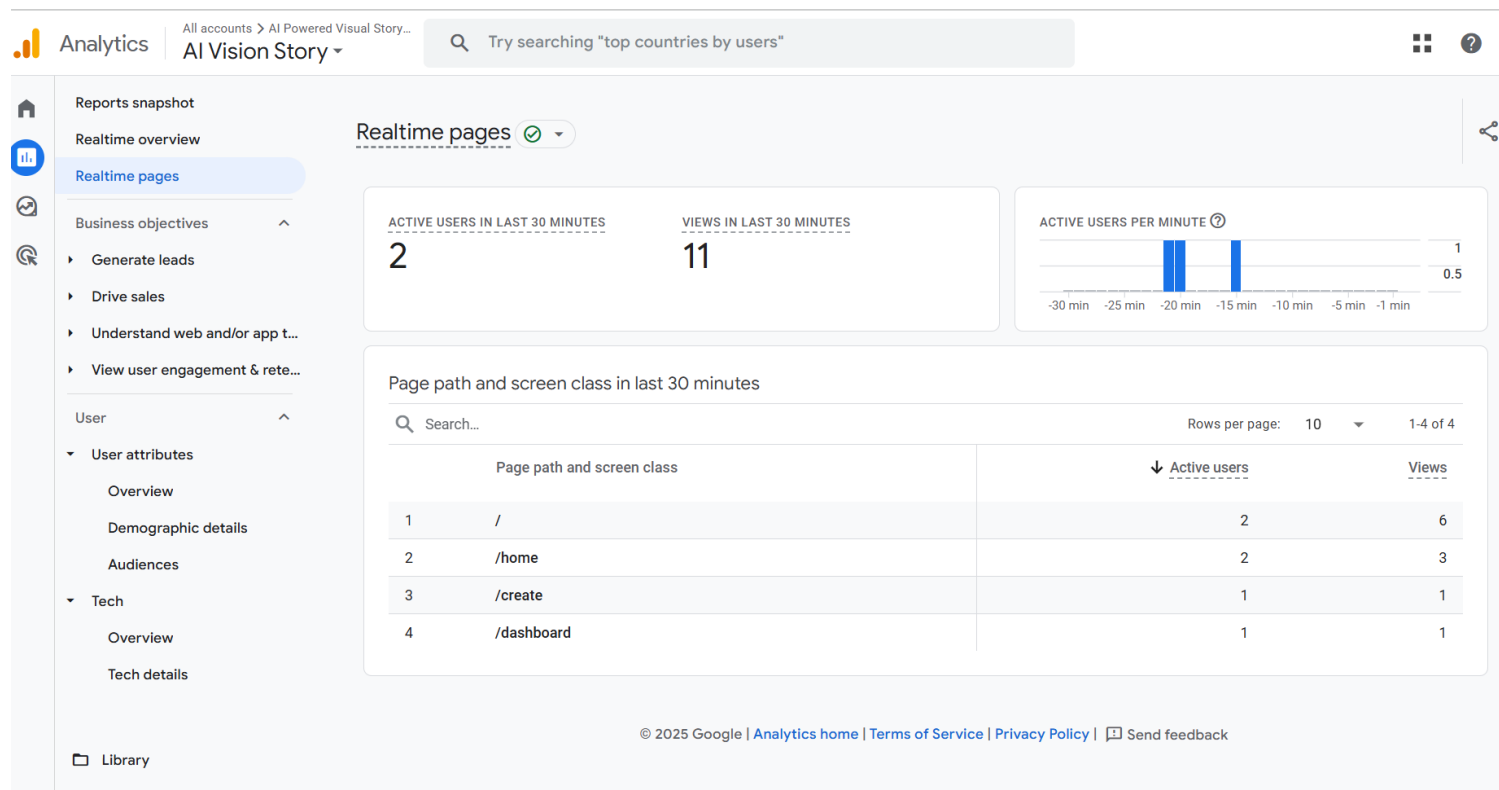
Google Analytics

We implemented a comprehensive Google Analytics 4 (GA4) strategy covering both the client-side (Frontend) and server-side (Backend) of the AI Powered Journal application.

1. Client-Side (Frontend): We utilized the react-ga4 library to initialize GA with our Measurement ID (G-4T4JT5XFMY). We created a utility module analytics.js to abstract the initialization and tracking logic. In App.js, we added a useEffect hook to automatically track page views whenever the user navigates between different views (e.g., Dashboard, Create Entry, Books). This ensures we capture the user's journey through the application.

2. Server-Side (Backend): Since the actual AI generation happens on the server, client-side tracking wouldn't capture the details of the API interactions reliably. We implemented the Measurement Protocol using axios in a backend analytics.js module. This allows us to send events directly to GA4 from our Node.js/Express server. We track critical events like vision_api_call and gemini_api_call, passing relevant metadata such as the number of images processed, the model used, and the requested tone/perspective.

Results in console:



Results in console:

analytics.google.com/analytics/web/#/a376389349p514762287/realtime/overview?params=_u...nav%3Dmaui&collectionId=user

Analytics AI Vision Story

Try searching "top countries by users"

Reports snapshot

Realtime overview

Realtime pages

Business objectives

Generate leads

Drive sales

Understand web and/or app t...

View user engagement & rete...

User

User attributes

Overview

Demographic details

Audiences

Tech

Overview

Tech details

Library

Event count by Event name

#1 page_view

11

45.83%

EVENT NAME	EVENT COUNT
page_view	11
gemini_api_call	3
vision_api_call	3
scroll	2
session_start	2
user_engagement	2

1 - 6 of 7

Key events by Event name

#1 -

No data available

EVENT NAME	KEY EVENTS
	No data available

Active users by User property

#1 -

No data available

USER PROPERTY	ACTIVE USERS
	No data available

© 2025 Google | Analytics home | Terms of Service | Privacy Policy | Send feedback

Results in console:

analytics.google.com/analytics/web/#/a376389349p514762287/realtime/overview?params=_u..nav%3Dmaui&collectionId=user

Analytics | All accounts > AI Powered Visual Story... | AI Vision Story

Try searching "top countries by users"

Reports snapshot | Realtime overview

Realtime overview

Realtime pages

Business objectives

- Generate leads
- Drive sales
- Understand web and/or app t...
- View user engagement & rete...

User

- User attributes
 - Overview
 - Demographic details
 - Audiences
- Tech
 - Overview
 - Tech details

Library

Event count by Event name

← gemini_api_call 3

EVENT PARAMETER KEY	EVENT COUNT
line_count	3
model	3
perspective	3
status	3
tone	3

1 - 5 of 5

Key events by Event name

#1 -

No data available

EVENT NAME	KEY EVENTS
------------	------------

No data available

Active users by User property

#1 -

No data available

USER PROPERTY	ACTIVE USERS
---------------	--------------

© 2025 Google | Analytics home | Terms of Service | Privacy Policy | Send feedback

Results in console:

analytics.google.com/analytics/web/#/a376389349p514762287/realtime/overview?params=_u..nav%3Dmaui&collectionId=user

Analytics | All accounts > AI Powered Visual Story... | AI Vision Story

Try searching "top countries by users"

Reports snapshot | Realtime overview

Realtime overview

Realtime pages

Business objectives

- Generate leads
- Drive sales
- Understand web and/or app t...
- View user engagement & rete...

User

- User attributes
 - Overview
 - Demographic details
 - Audiences
- Tech
 - Overview
 - Tech details

Library

Event count by Event name

← vision_api_call 3

EVENT PARAMETER KEY	EVENT COUNT
image_count	3
status	3

1 - 2 of 2 < >

Key events by Event name

#1 -

No data available

EVENT NAME	KEY EVENTS
------------	------------

No data available

Active users by User property

#1 -

No data available

USER PROPERTY	ACTIVE USERS
---------------	--------------

No data available

© 2025 Google | Analytics home | Terms of Service | Privacy Policy | Send feedback

Google Cloud Logging

We implemented Google Cloud Logging in the backend to capture detailed operational logs for all API requests made to external services such as Google Vision API and Google Gemini API. This setup allows the backend to produce structured, timestamped logs that are automatically collected and stored in Google Cloud's Logging system.

The setup uses the official `@google-cloud/logging` SDK, which provides a direct integration between the Node.js Express server and Google Cloud Logging.

A dedicated logging module (`cloudLogger.js`) was created to centralize the logic for writing logs. This file initializes the Google Cloud Logger, defines a log stream (`ai-vision-story-api`), and exposes a `writeLog()` function that accepts structured JSON objects

Results in console:

The screenshot displays the Google Cloud Logs Explorer interface. At the top, the Google Cloud logo and the project name 'AIPoweredJournal' are visible. A search bar contains the text 'Search (/) for resources, docs, products, and more'. The main section is titled 'Logs Explorer' and shows a search for 'ai-vision-story-api'. Below the search bar, there are filters for 'All resources', 'All log names', 'All severities', and 'Correlate by'. A timeline view shows a bar chart with a peak around 14:15. Below the timeline, there are 5 results. The first result is expanded, showing a log entry with the following details:

- Severity: INFO
- Time: 2025-12-01 14:14:22.259
- Summary: Vision API completed
- Log entry details:
 - insertId: ".....BBobhnxG7bPXLt15c4qQo"
 - jsonPayload: {
 - duration_ms: 947
 - message: "Vision API completed"
 - severity: "INFO"
 - success: true
 - logName: "projects/aipoweredjournal/logs/ai-vision-story-api"
 - receiveTimestamp: "2025-12-01T22:14:22.360134327Z"
 - resource: {2}
 - severity: "INFO"
 - timestamp: "2025-12-01T22:14:22.259000062Z"

The second result is partially visible below the first one, showing '2025-12-01 14:14:22.259 Gemini API called'.

Results in console:

The screenshot displays the Google Cloud Logs Explorer interface. At the top, the Google Cloud logo and the project name 'AIPoweredJournal' are visible. A search bar contains the text 'Search (/) for resources, docs, products, and more'. Below this, the 'Logs Explorer' section shows a search for 'ai-vision-story-api' under 'Project logs'. The interface includes filters for 'All resources', 'All log names', 'All severities', and 'Correlate by'. A 'Run query' button is present. The search results are displayed in a table with columns for 'SEVERITY', 'TIME', and 'SUMMARY'. The first result is '2025-12-01 14:14:22.259 Gemini API called'. Below the table, a detailed log entry is shown, including fields like 'insertId', 'jsonPayload', 'message', 'prompt_length', 'severity', 'logName', 'receiveTimestamp', 'resource', and 'timestamp'. A timeline view at the top of the results section shows a bar chart of log activity over time, with a peak around 14:15. The language is set to 'LQL'.

Google Cloud AIPoweredJournal Search (/) for resources, docs, products, and more Search

Logs Explorer Query library Share link Preferences Last 30 minutes PST Run query Show query

Project logs "ai-vision-story-api" Finds everything

All resources All log names All severities Correlate by

1 "ai-vision-story-api"

Example queries Query language guide Language: LQL

Timeline

5 results

SEVERITY	TIME	SUMMARY
i	2025-12-01 14:14:22.259	Gemini API called
i	2025-12-01 14:14:35.733	Gemini API success
i	2025-12-01 15:07:22.651	Vision API called

Explain this log entry Copy Expand nested fields Hide log summary

```
{
  insertId: ".....FBobhnxG7bPXLt15c4qQo"
  jsonPayload: {
    message: "Gemini API called"
    prompt_length: 336
    severity: "INFO"
  }
  logName: "projects/aipoweredjournal/logs/ai-vision-story-api"
  receiveTimestamp: "2025-12-01T22:14:22.360134327Z"
  resource: {2}
  severity: "INFO"
  timestamp: "2025-12-01T22:14:22.259000062Z"
}
```

Results in console:

The screenshot displays the Google Cloud Logs Explorer interface. At the top, the Google Cloud logo and project name 'AIPoweredJournal' are visible. A search bar contains the text 'Search (/) for resources, docs, products, and more'. Below this, the 'Logs Explorer' section shows a search for 'ai-vision-story-api'. The interface includes a 'Project logs' dropdown, a search bar with the query 'ai-vision-story-api', and a 'Run query' button. A 'Show query' toggle is also present. The search results are displayed in a table with columns for SEVERITY, TIME, and SUMMARY. The results show a sequence of events related to the Vision API and Gemini API. A timeline view is also available, showing the distribution of log entries over time. The timeline shows a peak in activity around 14:14:21. The table below the timeline lists the following results:

SEVERITY	TIME	SUMMARY
Info	2025-12-01 14:14:21.312	Vision API called
Info	2025-12-01 14:14:21.332	{"logging.googleapis.com/diagnostic":{"_":}}
Info	2025-12-01 14:14:22.259	Vision API completed
Info	2025-12-01 14:14:22.259	Gemini API called
Info	2025-12-01 14:14:35.733	Gemini API success

Summary

If you had and problems that didn't work, show them (the results if any) and discuss why you think it didn't work

- Pinterest login failed-Pinterest requires the app to be verified before it enables OAuth login for users. Because our app hasn't yet completed Pinterest's verification process, the Pinterest login flow is blocked.
- Instagram posting limitations-Instagram's APIs restrict what third-party web apps can publish. In our case we found we cannot programmatically post both image and caption from the web app the way we wanted — Instagram's web-facing APIs are purposefully limited for security and policy reasons (they allow some read/display features but not full content publishing from arbitrary web apps).

Summary

Discuss how you might improve this app

- Increase media limits — allow more than 20 images with client-side pagination/virtualization to keep the UI performant.
- Pinterest as a media source (post-verification) — once verified, integrate Pinterest import to leverage its wide range of photos.

Summary

Discuss what you learned from making this app- what were the challenges, what you learned, how you can help others.

- Finding the right Gemini model-We experimented with several Gemini variants (1.2 Pro, 1.5 Turbo, etc.). Through iterative testing we found Gemini 2.5 Pro produced the narrative for image-driven stories.
- Connecting and designing Firestore queries-Designing the Firestore schema and queries (creating root collections, sub-collections, ensuring atomic writes, and removing orphaned media) required learning best practices for nested data and security rules.
- A key challenge occurred when deploying to GCP, where App Engine could not be recreated due to leftover metadata from a previously deleted instance, ultimately requiring migration to a new project. This experience emphasized the importance of understanding cloud resource behavior and handling platform-specific errors. These insights can help others anticipate similar issues and manage their deployments more smoothly.