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

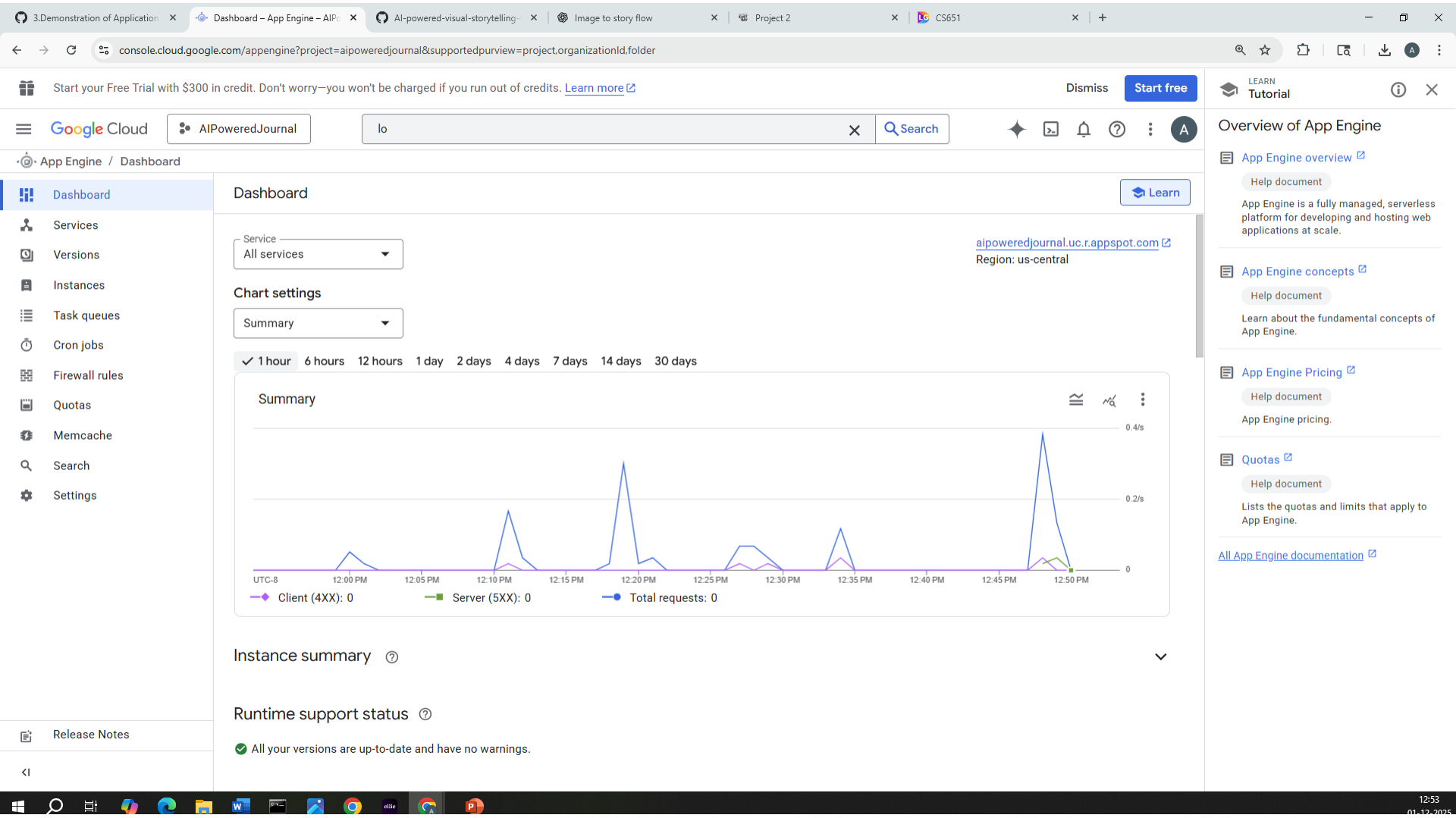
Deployment

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



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 Querries

- 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.

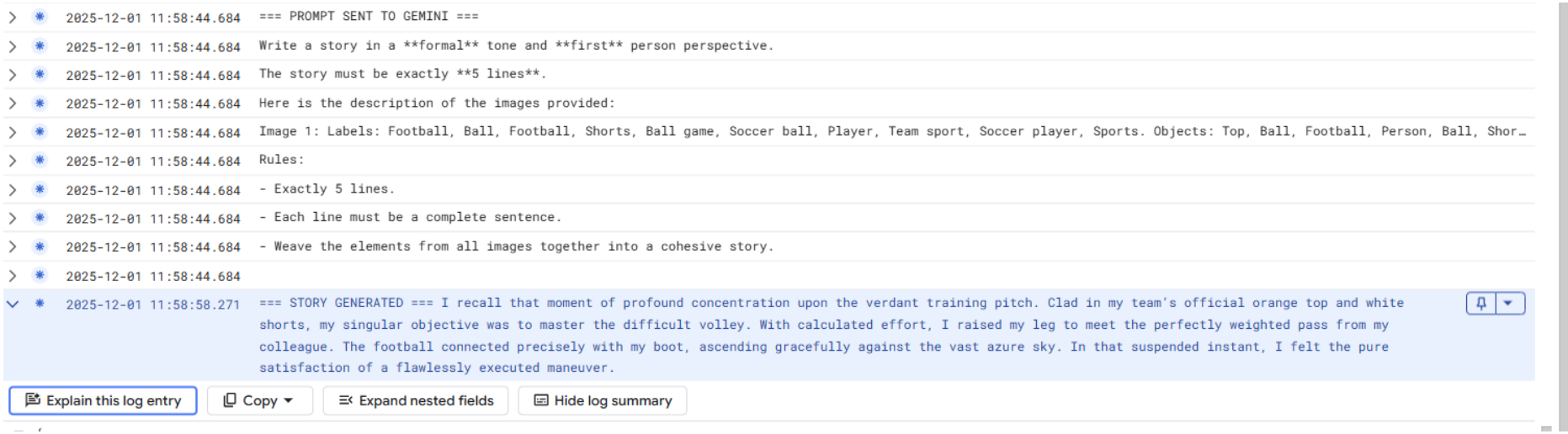
Vision Api output

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



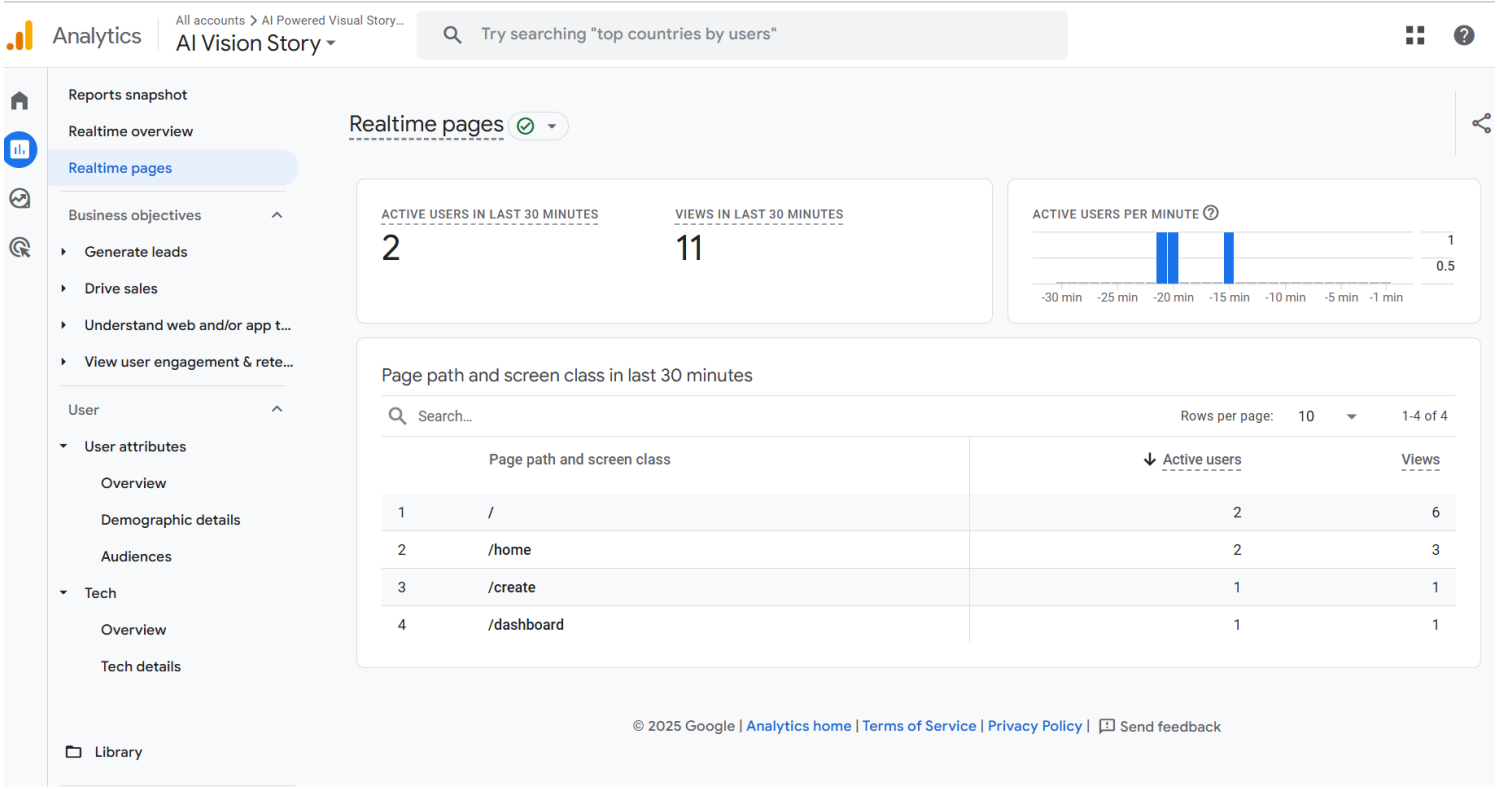
Google Analytics

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.

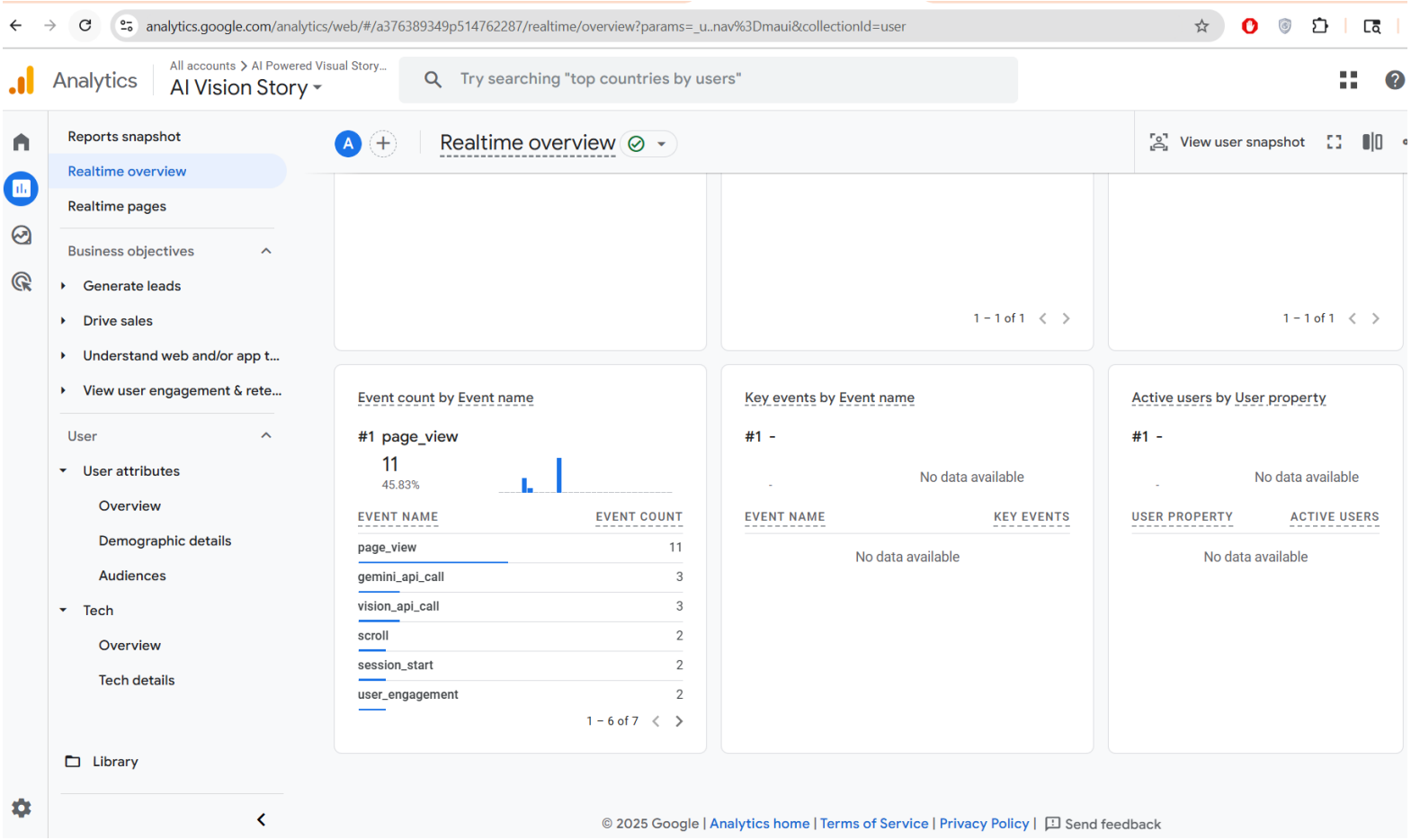
1. Client-Side (Frontend): We utilized the react-ga4 library to initialize GA with our Measurement ID (G-4T4JT5XFMY). We created a utility module analytics.js to abstract the initialization and tracking logic. In App.js, we added a useEffect hook to automatically track page views whenever the user navigates 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.

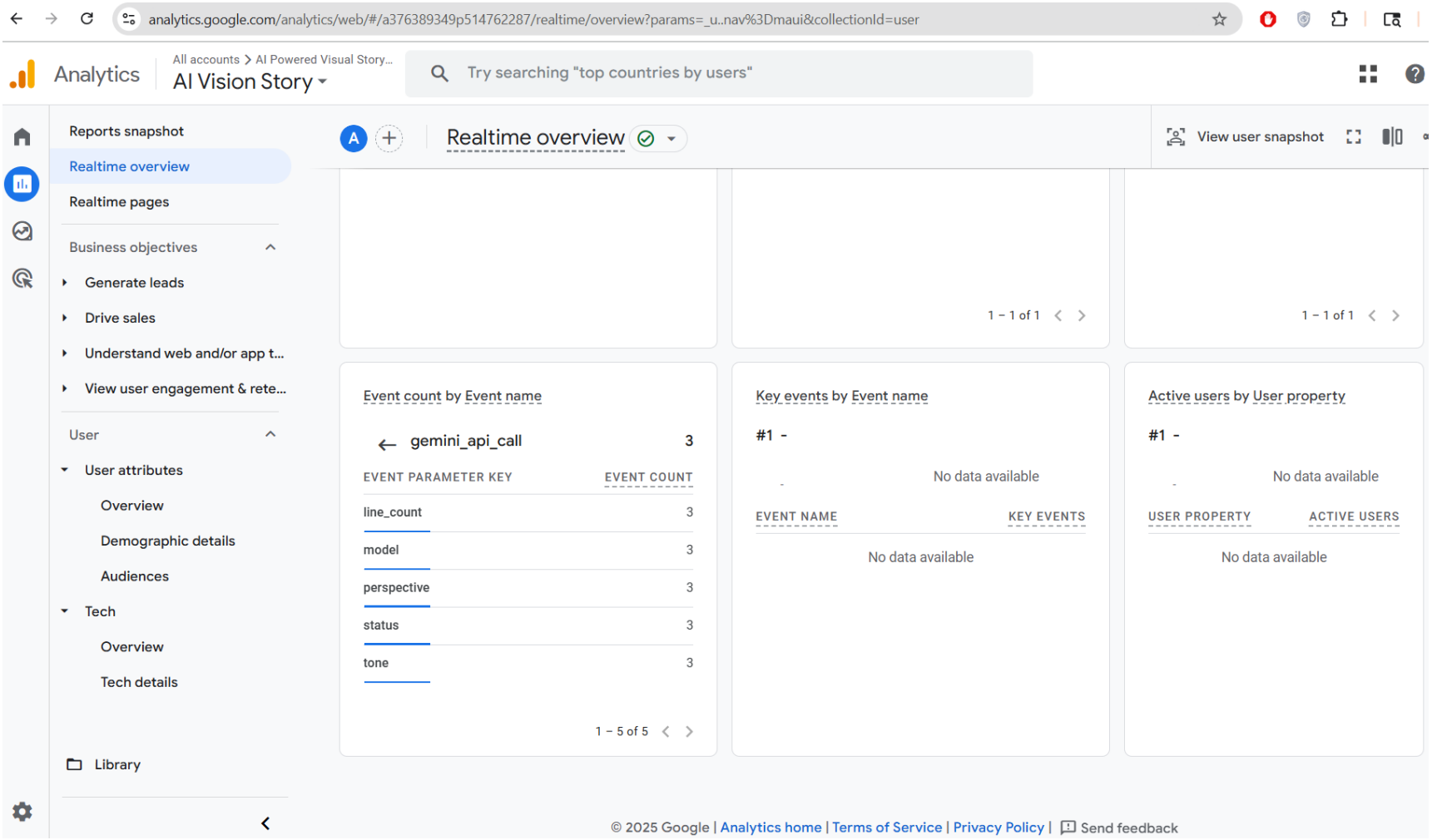
Results in console:



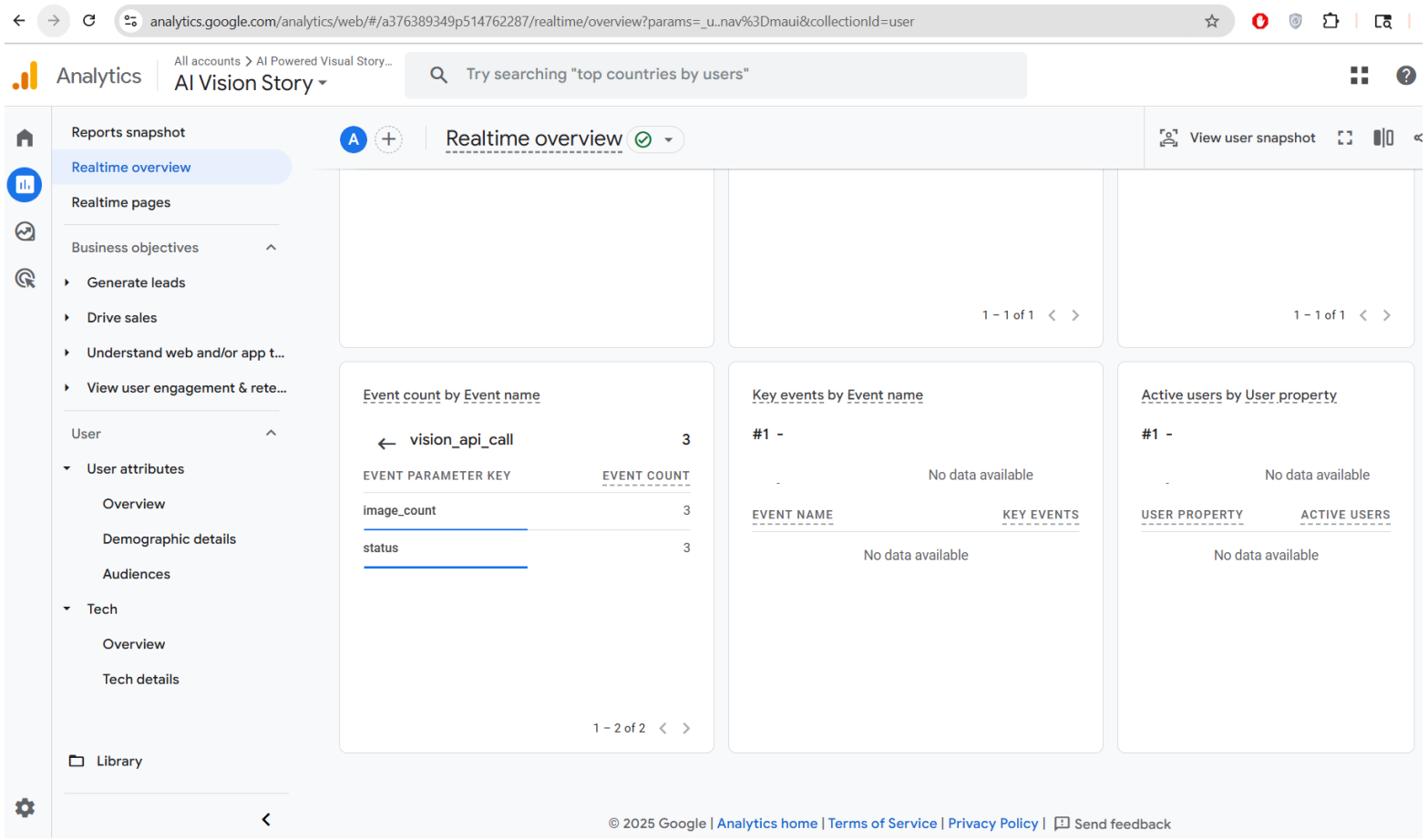
Results in console:



Results in console:



Results in console:



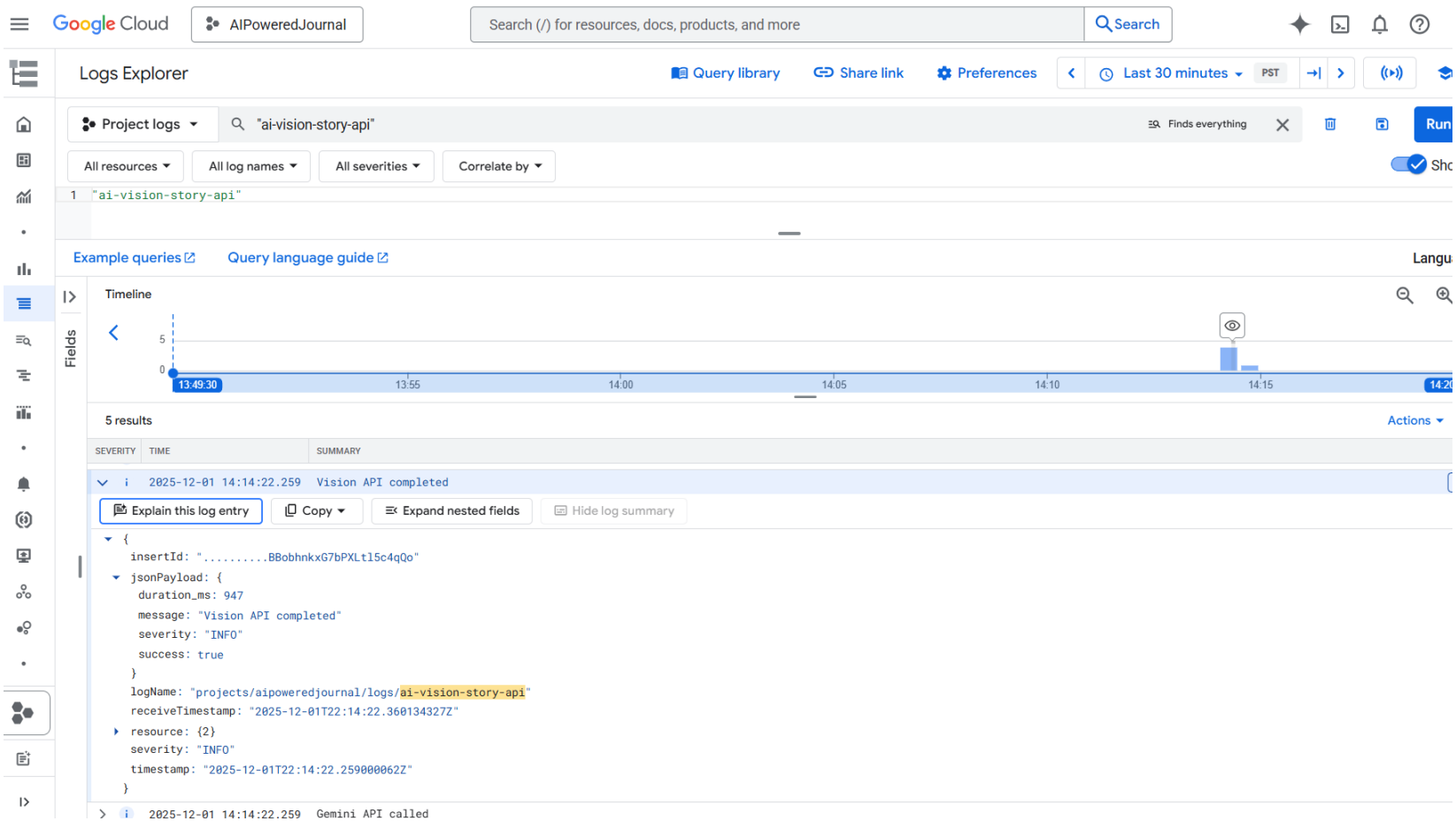
Google Cloud Logging

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.

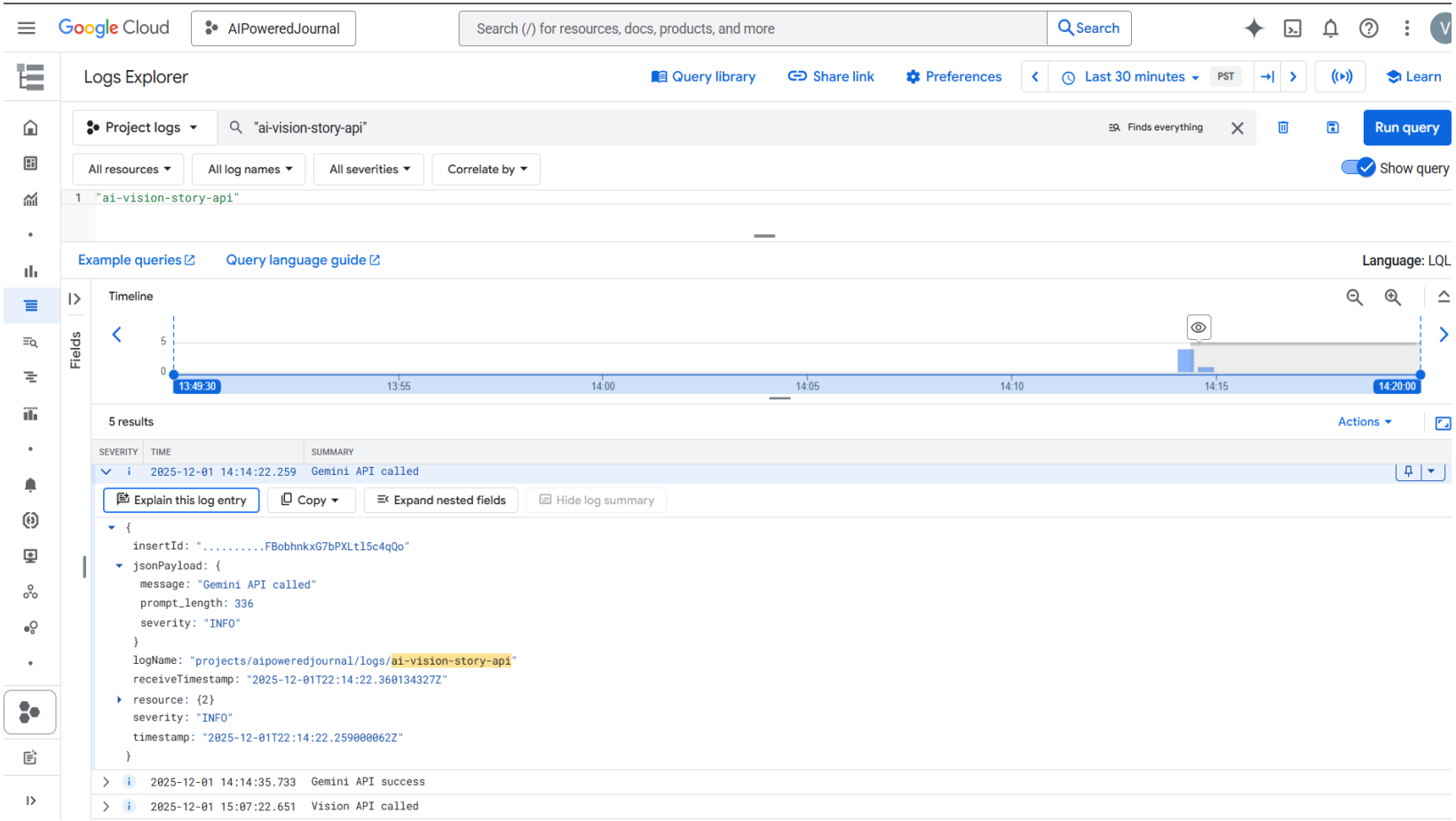
The setup uses the official @google-cloud/logging SDK, which provides a direct integration between the Node.js Express server and Google Cloud Logging.

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

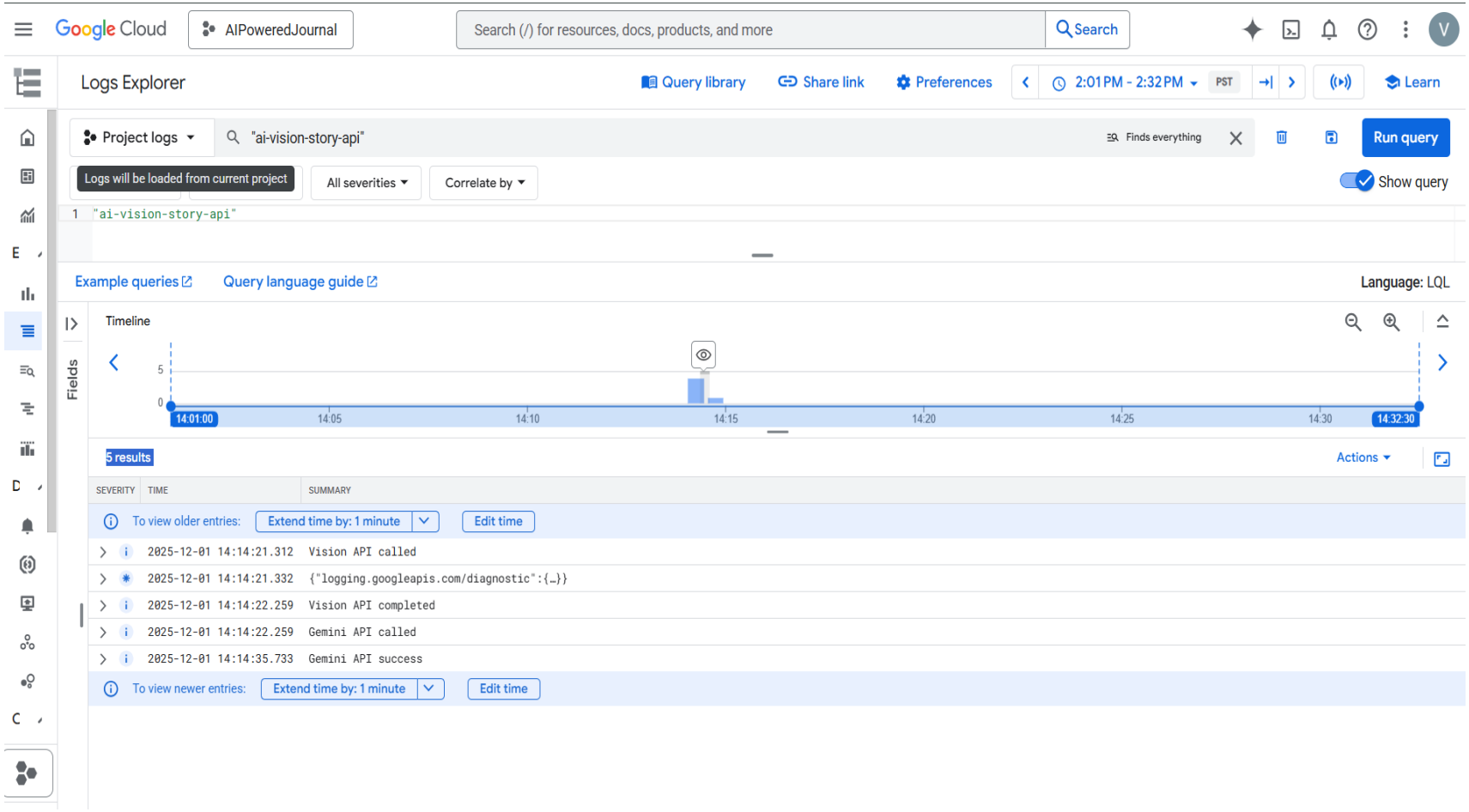
Results in console:



Results in console:



Results in console:



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.