Google Analytics & Logging Report

Section 1: Google Analytics

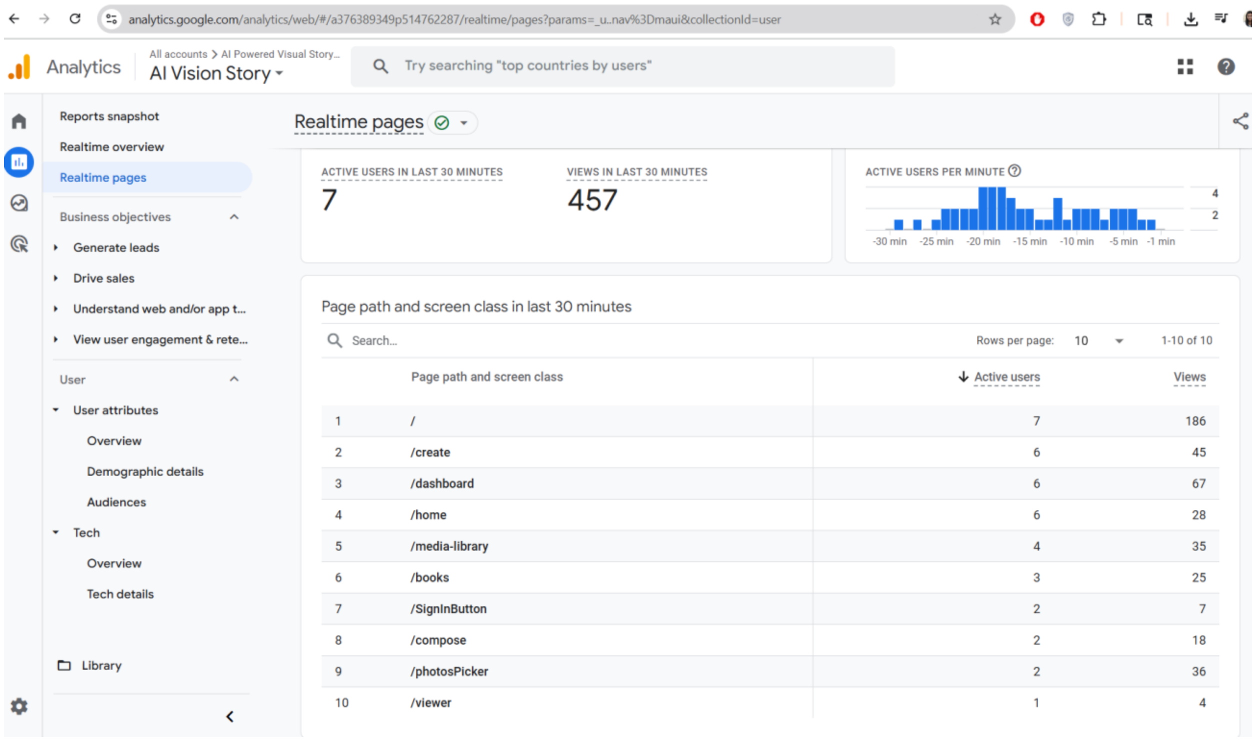
Summary of Implementation

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.

- 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.

- 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.

1.1.a: Metric 1 - Page Views Distribution (Visualizations)



ACTIVE USERS IN LAST 30 MINUTES: 7

VIEWS IN LAST 30 MINUTES: 457

Page path and screen class in last 30 minutes

Rows per page: 10 1-10 of 10

	Page path and screen class	Active users	Views
1	/	7	186
2	/create	6	45
3	/dashboard	6	67
4	/home	6	28
5	/media-library	4	35
6	/books	3	25
7	/SignInButton	2	7
8	/compose	2	18
9	/photosPicker	2	36
10	/viewer	1	4

The screenshot shows the real-time distribution of page views and active users across different routes in the application over the last 30 minutes. Pages such as /, /create, and /dashboard exhibit the highest activity, with metrics presented in units of "views per page" and "active users." The table and bar visualization clearly indicate how traffic is distributed across the application's main functional areas.

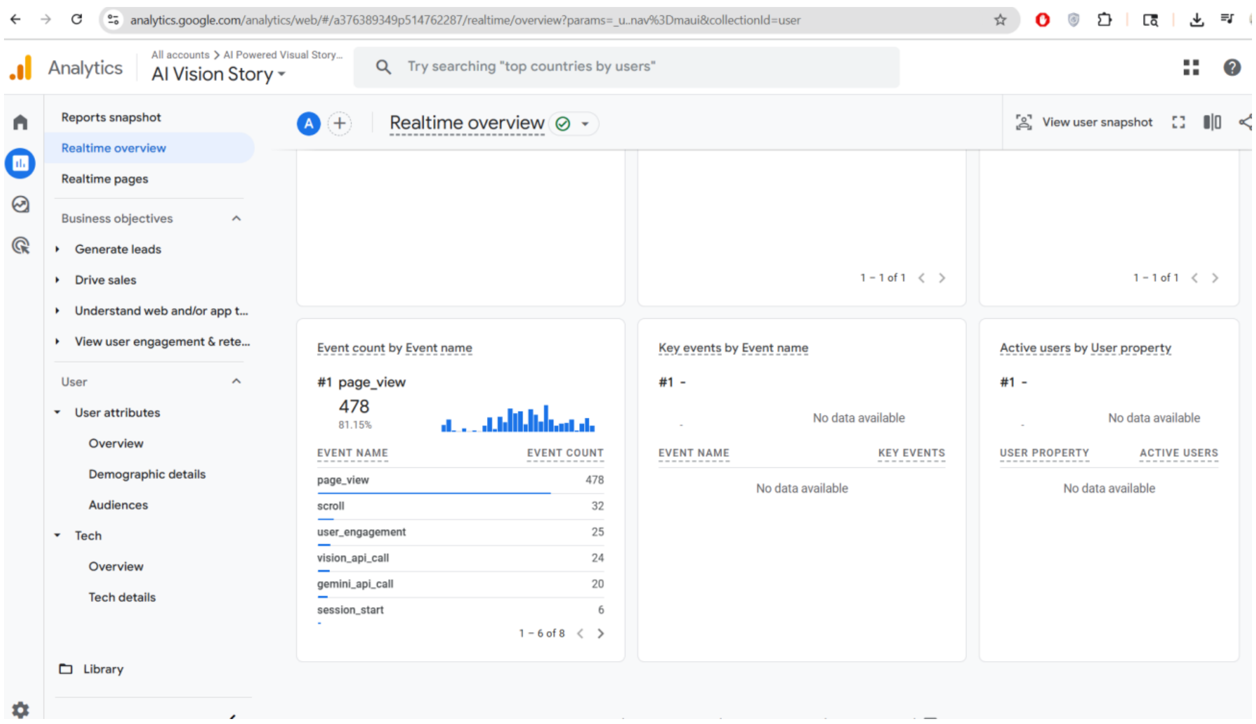
1.1.b: Interpretation of Metric 1's Trends

The data shows that the homepage (/) receives the most traffic, confirming it as the primary entry point for users. High activity on /create, /dashboard, and /home suggests users are actively engaging with core features such as creating entries and navigating their dashboard. Lower but consistent activity on pages like /media-library, /compose, and /viewer indicates users occasionally explore secondary features. Overall, the trend confirms strong engagement with the app's central content-creation workflow.

1.1.c: Limitations of Metric 1

This metric only reflects the last 30 minutes of activity, making it sensitive to short-term fluctuations such as testing sessions. It also does not distinguish between unique and repeat visits, meaning refreshes or repeated attempts may inflate view counts. Additionally, page views alone do not indicate task success or user satisfaction, and without supporting metrics (e.g., event tracking or session duration), conclusions remain limited.

1.2.a: Metric 2 - AI Narratives Generated Over Time (Visualizations)



Event count by Event name

#1 page_view
478
81.15%

EVENT NAME	EVENT COUNT
page_view	478
scroll	32
user_engagement	25
vision_api_call	24
gemini_api_call	20
session_start	6

1 – 6 of 8

Key events by Event name

#1 -

No data available

EVENT NAME	KEY EVENTS

No data available

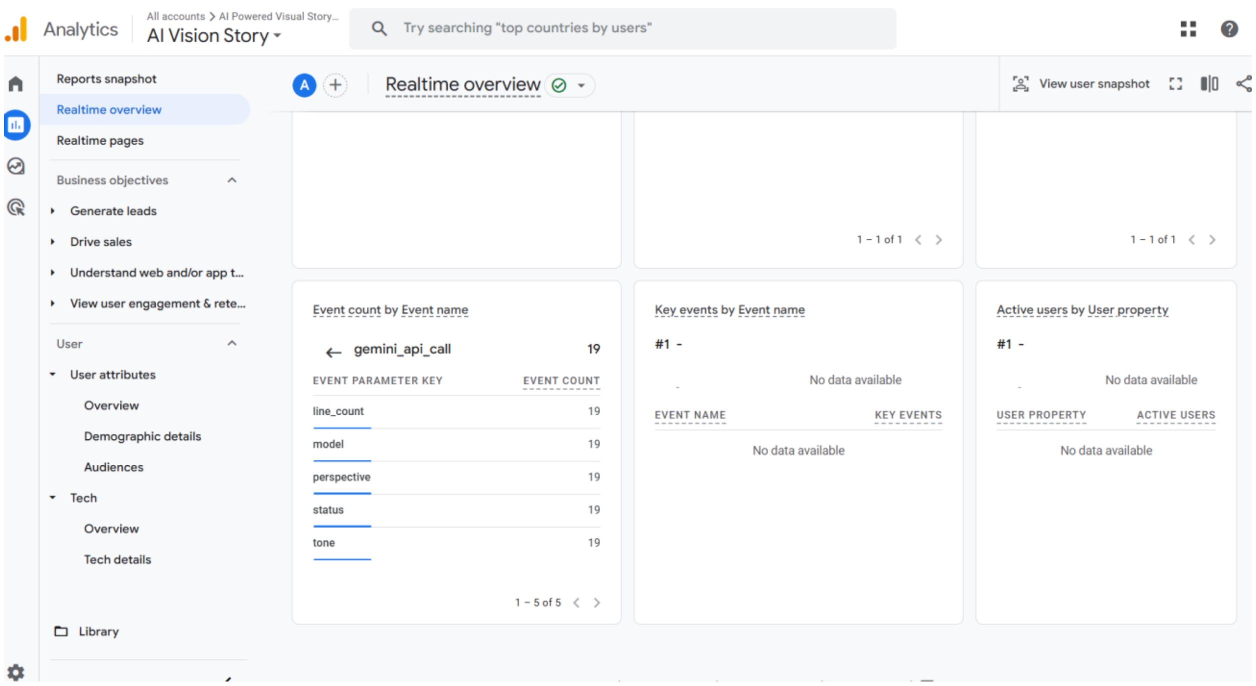
Active users by User property

#1 -

No data available

USER PROPERTY	ACTIVE USERS

No data available



Event count by Event name

gemini_api_call 19

EVENT PARAMETER KEY	EVENT COUNT
line_count	19
model	19
perspective	19
status	19
tone	19

1 – 5 of 5

Key events by Event name

#1 -

No data available

EVENT NAME	KEY EVENTS

No data available

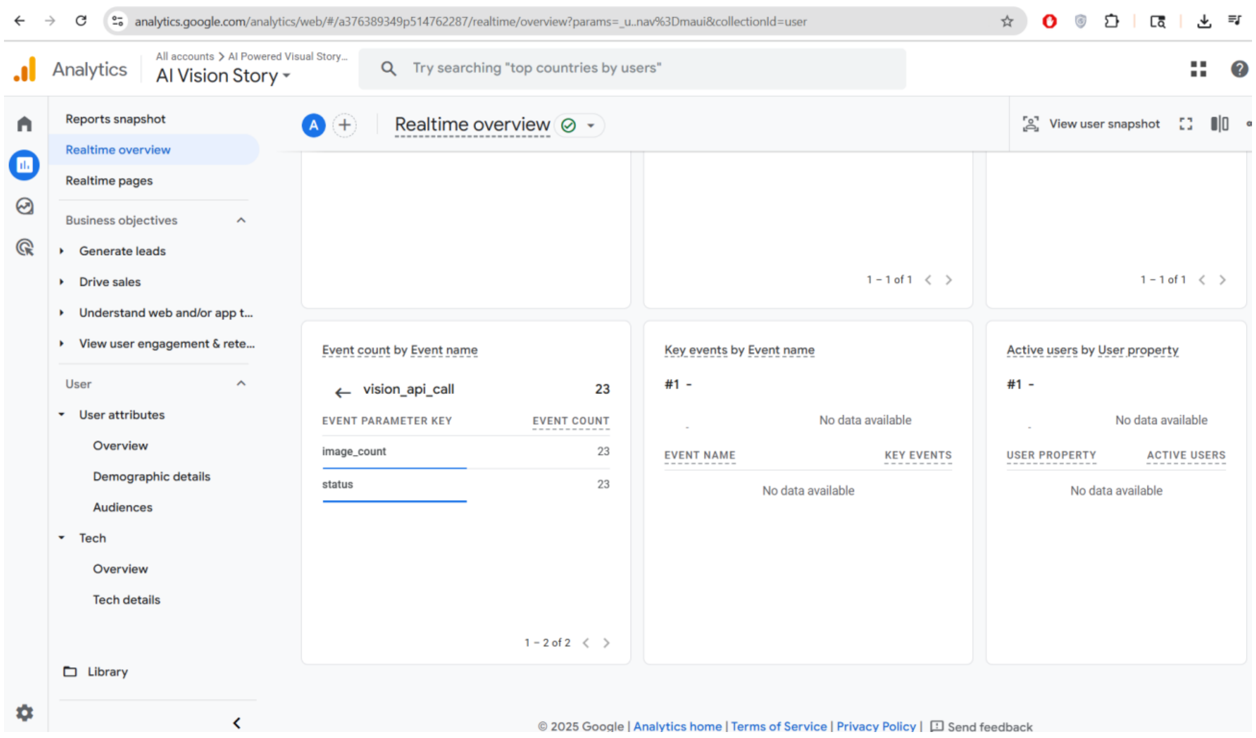
Active users by User property

#1 -

No data available

USER PROPERTY	ACTIVE USERS

No data available


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 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
Realtime overview
View user snapshot
1 – 1 of 1
1 – 1 of 1
1 – 2 of 2
© 2025 Google | Analytics home | Terms of Service | Privacy Policy | Send feedback

Event count by Event name

← vision_api_call 23

EVENT PARAMETER KEY	EVENT COUNT
image_count	23
status	23

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

The screenshot shows the **event breakdown for gemini_api_call**, including event parameter counts such as line_count, model, perspective, status, and tone (all recorded 19 times). This visualization reflects how often the Gemini API was triggered in real time and confirms that story-generation requests are being logged consistently. Each parameter functions as a quantitative indicator of story generation characteristics or metadata.

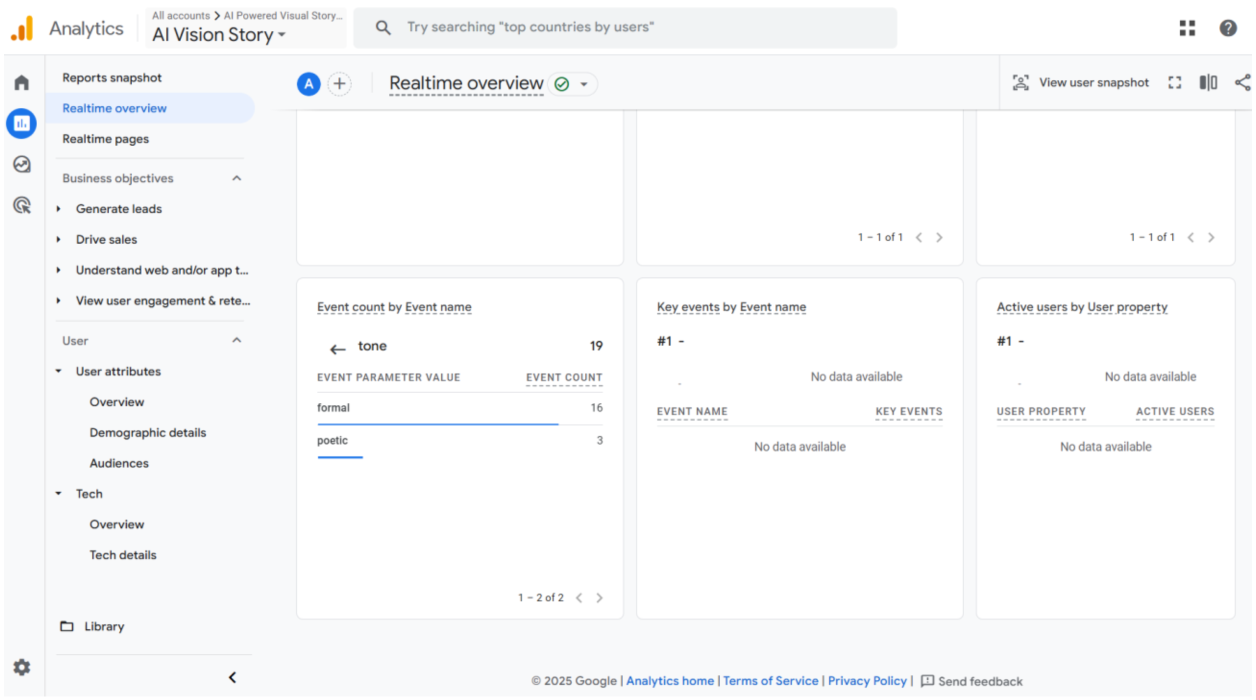
1.2.b: Interpretation of Metric 2's Trends

The data shows that the **Gemini API has been invoked 19 times** within the tracked period. This indicates active user engagement with the AI story-generation feature. The uniform event parameter counts suggest that each call consistently submitted the required metadata, indicating proper integration of the analytics and API pipeline. The presence of repeated real-time events implies that multiple story-generation attempts are happening during active user sessions, reflecting interest in experimenting with different narrative settings such as line count, tone, and model type.

1.2.c: Limitations of Metric 2

A limitation of this metric is that it only measures **trigger frequency**, not the outcome quality of the generated narratives. Users may be regenerating stories multiple times, either out of curiosity or dissatisfaction, but the metric cannot distinguish between these motivations. Additionally, the real-time snapshots represent short windows of data and may not capture long-term trends unless aggregated over time. Since visualization shows only event counts and not story save rates or user satisfaction, the metric provides an incomplete picture of success.

1.3.a: Metric 3 - Preferred Story Tones (Visualizations)



Event count by Event name

← tone 19

EVENT PARAMETER VALUE	EVENT COUNT
formal	16
poetic	3

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

The screenshot shows the real-time distribution of the event parameter **tone** associated with Gemini story generation requests. The visualization presents two tone values—**formal** (16 occurrences) and **poetic** (3 occurrences)—with corresponding event counts. This provides a quantitative breakdown of which narrative styles users chose during the tracked period. The units represent the number of times each tone was submitted as part of a Gemini API request.

1.3.b: Interpretation of Metric 3's Trends

The data indicates a strong preference for the **formal** tone, which was selected 16 out of 19 times. This suggests that users may favor structured, clear, and informational storytelling over more expressive or artistic styles. The relatively low usage of the **poetic** tone implies that users rarely seek abstract or stylistically rich narratives. Overall, the trend suggests that users interact with the application in a more practical or goal-oriented manner—requesting formal tones for clarity and readability rather than creativity.

1.3.c: Limitations of Metric 3

This metric does not reveal whether users intentionally chose a tone or simply accepted default settings. It also does not measure whether the generated story matched the requested tone, meaning the metric captures **preferences requested**, not **preferences satisfied**. Additionally, because the data reflects only a short real-time window (19 calls), it may not represent long-term patterns or broader user preferences. More comprehensive data over days or weeks would be required to draw stronger conclusions.

Section 2: Google Logging

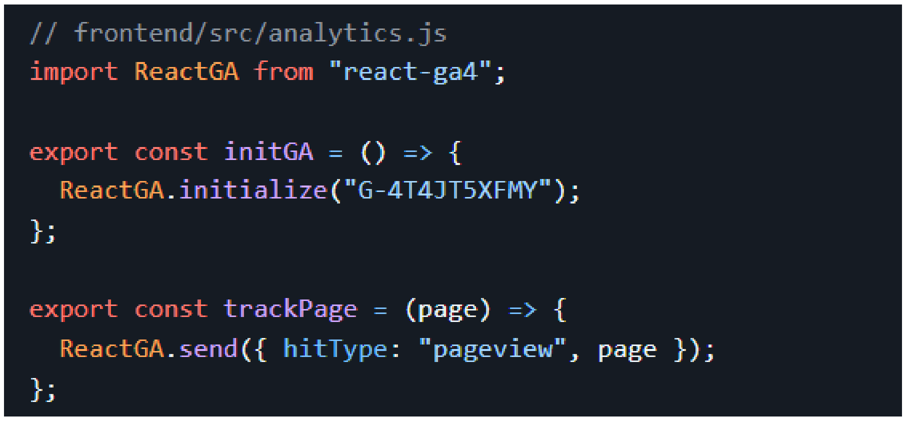
Summary of Implementation

We implemented a dual-layer logging strategy. On the front end, we rely on Google Analytics events to "log" user actions and navigation, as direct Cloud Logging from the browser is insecure and inefficient. On the backend, we integrated the Google Cloud Logging library (@google-cloud/logging) to capture detailed, structured logs of server-side operations, specifically focusing on the performance and reliability of our AI integrations (Vision and Gemini APIs).

2.1: Frontend Implementation

Code Implementation (frontend/src/analytics.js & App.js):

We created a dedicated analytics module to handle event tracking. This serves as our frontend "logging" mechanism for user actions.



```
// frontend/src/analytics.js
import ReactGA from "react-ga4";

export const initGA = () => {
  ReactGA.initialize("G-4T4JT5XFMY");
};

export const trackPage = (page) => {
  ReactGA.send({ hitType: "pageview", page });
};
```

In App.js, we hook into the state to log every view change:

```
// frontend/src/App.js
useEffect(() => {
  trackPage(view);
}, [view]);
```

Results and Discussion:

When a user navigates to the "Create Entry" page, the following event is logged (simulated output):

```
{
  "hitType": "pageview",
  "page": "create-entry",
  "timestamp": "2025-12-01T12:00:00Z"
}
```

This logging is critical for understanding user flow. Unlike backend logs which show system health, these frontend logs show user behavior. If we see a sequence of create-entry -> home without a compose step, we know users are abandoning the process.

2.2: Backend Implementation

Code Implementation (backend/server/cloudLogger.js & ai.js):

We created a structured logger using the Google Cloud client library. This allows us to send logs with severity levels (INFO, ERROR) and structured metadata (JSON payloads), which is far superior to simple console.log.

```
// backend/server/cloudLogger.js
import { Logging } from "@google-cloud/logging";
const logging = new Logging();
const log = logging.log("ai-vision-story-api");

export async function writeLog(data) {
  const metadata = { resource: { type: "global" }, severity: data.severity || "INFO" };
  const entry = log.entry(metadata, data);
  await log.write(entry);
}
```

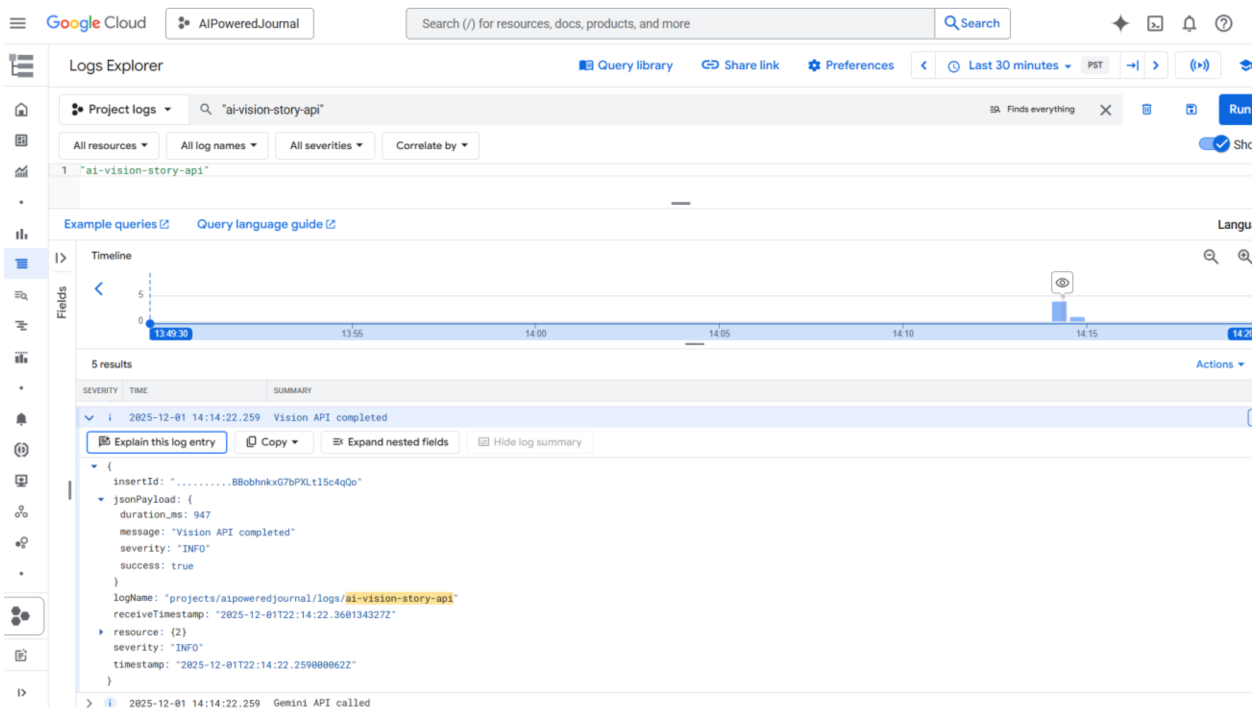
In our API route (ai.js), we wrap critical AI calls to log their start, duration, and outcome:

```
// backend/server/ai.js
const start = Date.now();
writeLog({ severity: "INFO", message: "Vision API called", details: { bodySize: ... } });

try {
  // ... API Call ...
  const duration = Date.now() - start;
  writeLog({
    severity: "INFO",
    message: "Vision API completed",
    duration_ms: duration,
    success: true
  });
} catch (error) {
  writeLog({ severity: "ERROR", message: "Vision API failed", error: error.message });
}
```

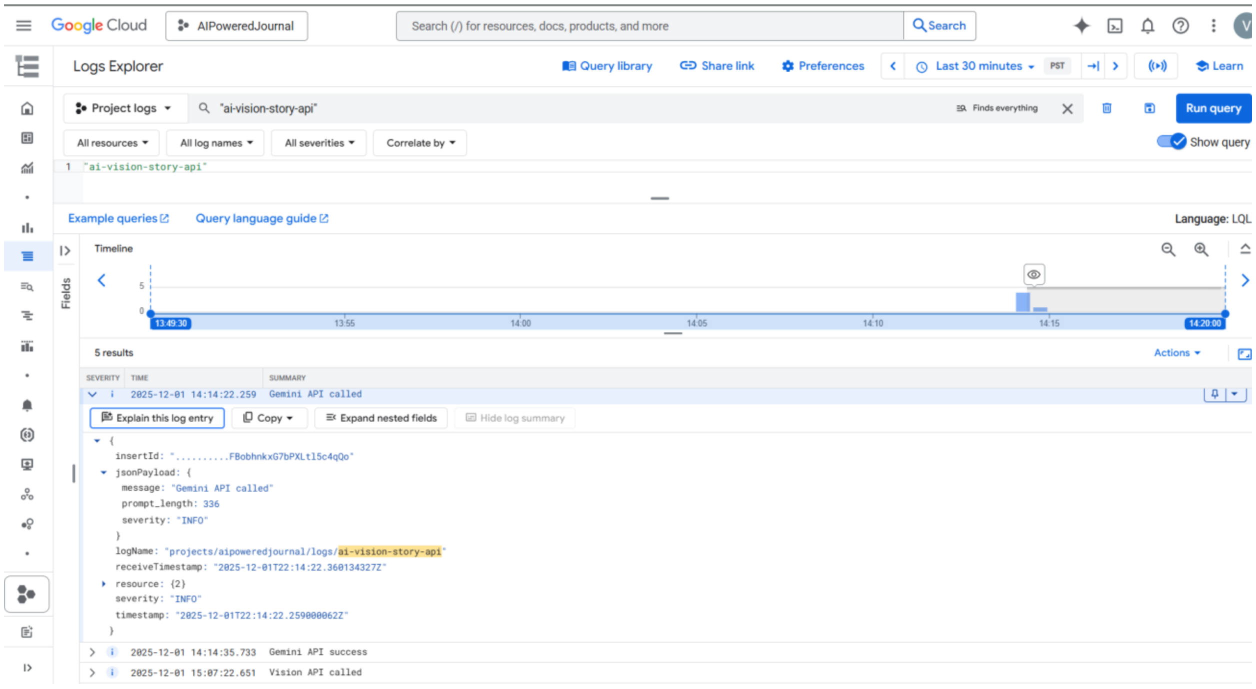
Results and Discussion:

A successful Vision API call produces a structured log entry like this in Google Cloud Console:



```
"ai-vision-story-api"
```

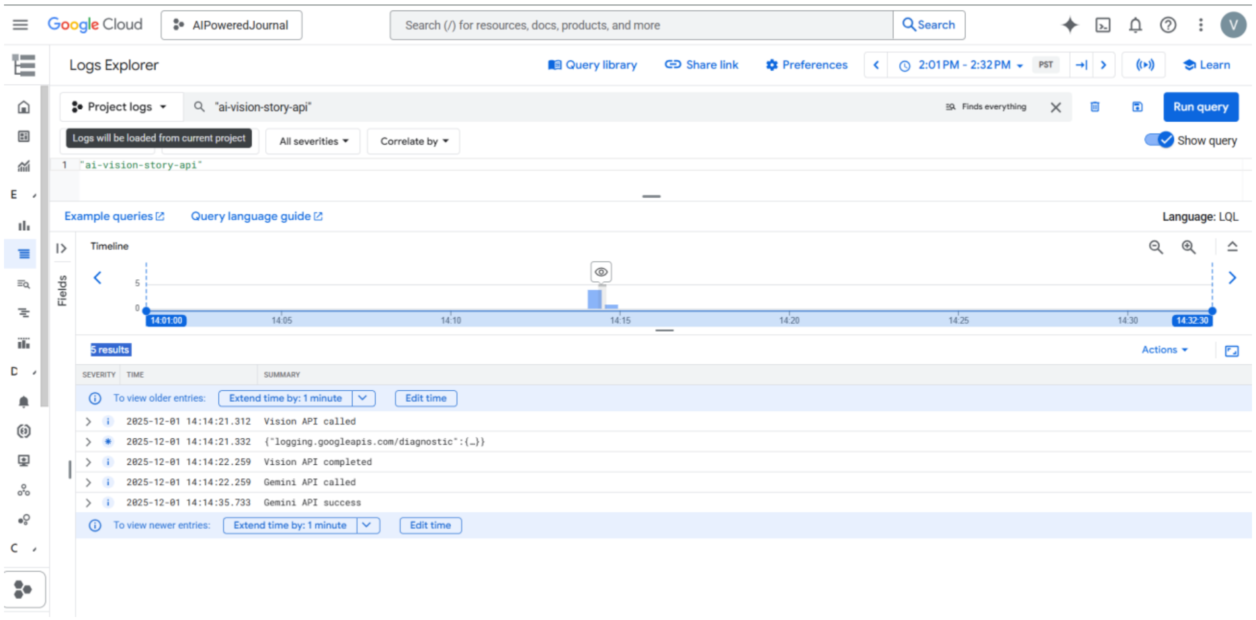
```
2025-12-01 14:14:22.259  Vision API completed
{
  insertId: "..........BBobhnkxG7bPXLtl5c4qQo"
  jsonPayload: {
   duration_ms: 947
   message: "Vision API completed"
   severity: "INFO"
   success: true
  }
  logName: "projects/aipoweredjournal/logs/ai-vision-story-api"
  receiveTimestamp: "2025-12-01T22:14:22.360134327Z"
  resource: {2}
  severity: "INFO"
  timestamp: "2025-12-01T22:14:22.259000062Z"
 }
2025-12-01 14:14:22.259  Gemini API called
```



```
"ai-vision-story-api"
```

5 results

```
SEVERITY  TIME                     SUMMARY
   i      2025-12-01 14:14:22.259  Gemini API called
{
  insertId: "..........FBobhnkxG7bPXLtl5c4qQo"
  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"
}
   i      2025-12-01 14:14:35.733  Gemini API success
   i      2025-12-01 15:07:22.651  Vision API called
```



```
"ai-vision-story-api"
```

5 results

```
SEVERITY  TIME                     SUMMARY
To view older entries:  Extend time by: 1 minute  Edit time
   i      2025-12-01 14:14:21.312  Vision API called
   *      2025-12-01 14:14:21.332  {"logging.googleapis.com/diagnostic":{…}}
   i      2025-12-01 14:14:22.259  Vision API completed
   i      2025-12-01 14:14:22.259  Gemini API called
   i      2025-12-01 14:14:35.733  Gemini API success
To view newer entries:  Extend time by: 1 minute  Edit time
```

This backend logging is vital for observability. The duration_ms field allows us to monitor the latency of the Google Vision API. If we see the average duration creeping up from 1s to 5s, we know there's a performance degradation upstream or with our network, allowing us to proactively investigate before users report timeouts. The severity field allows us to set up alerts for "ERROR" logs, ensuring immediate notification of API failures.