Homework 5: Firestore API

Due: See online for due date/time **Collaboration:** Individual/group Assignment **GenAI:** Limited (see below)

Scenario

You have been temporarily reassigned from the standard infrastructure team to a new social media project. The core feature is that users can create real-time interactive polls to engage with their followers before the polls expire and disappear.

This system utilizes serverless architecture. Your task is to build a fully automated polling API where a user creates a profile, launches a poll, and the system automatically tracks votes and handles expiration logic using background triggers. You must ensure that the system is robust enough to prevent duplicate voting and can handle high-concurrency updates to vote totals. This assignment demonstrates decoupling services where the API and the data-processing triggers act independently of each other to deliver the functionality.

Implementation

Architecture Overview You will be deploying a suite of Cloud Functions that handle the lifecycle of a polling application:

1. **API Main (HTTP Function):** A public endpoint to handle all RESTful requests (Users, Polls, Votes).
2. **Vote Processor (Firestore Trigger):** A function that fires when a new document is added to the votes collection to update aggregates.

Infrastructure

- **Database:** Initialize a Firestore database.
- **Service Accounts:** Create a new Service Account for each service and grant them the least privileges. Discuss your choice.
    - hw-5-api-agent (Permissions to read/write Firestore).
    - hw-5-trigger-agent (Permissions to update Firestore based on events).
- **Indexing:** You will be required to create a Firestore composite index to support specific filtered queries.

Functions
Function A: Polling API (HTTP Trigger)

- **Trigger:** HTTP (Allow unauthenticated invocations).
- **Logic:** Implement the following RESTful endpoints:
    - POST /users: Create a unique username or return a generated one.
    - POST /polls: Create a new poll with a question, options, and expiration.
    - POST /polls/{pollId}/vote: Cast a vote.

        - **Must check Firestore first to prevent duplicate votes by the same user.**
    - GET /polls/{pollId}: Retrieve current results for a specific poll.
    - GET /polls?owner={userId}&expiresBefore={timestamp}: Retrieve a list of the user's polls expiring within 24 hours.
        - When running a query that requires an index for the first time, Firestore will prompt you to make one via an index creation link in the error log for that call.
    - DELETE /polls/{pollId}: Remove a poll (Ensure only the creator can delete it).

Function B: Vote Counter (Firestore Trigger)

- **Trigger:** providers/cloud.firestore/eventTypes/document.create on the votes collection.
- **Logic:** * Whenever a new vote is submitted, this function must automatically increment the total_votes count in the corresponding poll document.
    - If a user attempts to vote on a poll past the expiration date, the function should mark the poll as closed.
    - The vote should **not** be counted in the final tally.

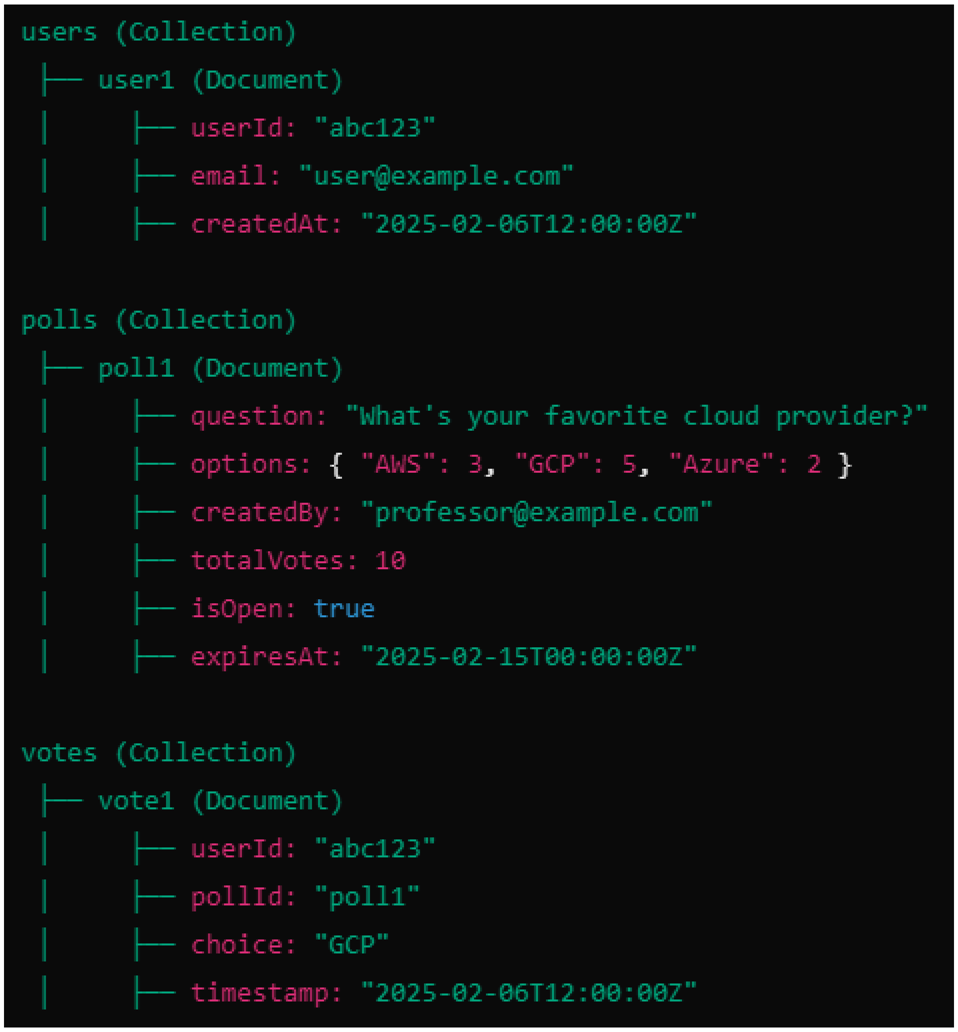
Data Structure (Suggested)

- **Users Collection:** username, created_at.
- **Polls Collection:** question, options_map, creator_id, total_votes, status (open/closed), expiration_timestamp.
- **Votes Collection:** voter_id, poll_id, option_selected, timestamp.

Extra Credit Challenge: Anonymous Voting

- Modify the API to allow voting without a userId.
- Track and prevent duplicate votes based on the **client IP address** instead.

o Example layout (feel free to use a different one also):



```
users (Collection)
 ├── user1 (Document)
 │      ├── userId: "abc123"
 │      ├── email: "user@example.com"
 │      ├── createdAt: "2025-02-06T12:00:00Z"

polls (Collection)
 ├── poll1 (Document)
 │      ├── question: "What's your favorite cloud provider?"
 │      ├── options: { "AWS": 3, "GCP": 5, "Azure": 2 }
 │      ├── createdBy: "professor@example.com"
 │      ├── totalVotes: 10
 │      ├── isOpen: true
 │      ├── expiresAt: "2025-02-15T00:00:00Z"

votes (Collection)
 ├── vote1 (Document)
 │      ├── userId: "abc123"
 │      ├── pollId: "poll1"
 │      ├── choice: "GCP"
 │      ├── timestamp: "2025-02-06T12:00:00Z"
```

What to Turn In

You may turn in :

1. A **3-5 minute video** (longer is allowed if needed, but no more than 10 minutes. Points will be deducted for videos in excess of 10 minutes) of you demonstrating the code, it running, and the outcome
    a. You can show and talk through slides, console, code, etc. the goal is to convince me you completed the assignment correctly. If I cannot understand what you did or see it finished and running I cannot give full points. If you could not finish the assignment, talk through the parts you completed and what you got stuck on.
    b. API testing and validation using whatever method you prefer. You must test every endpoint for full credit.

NOTE: You must include a discussion of how you used GenAI in your work if any was used.

Submission Instructions

1. **SUBMISSION:** Submit one of the following two options:
    a. **Option 1:** Your video as a **YouTube link** (make sure it is not private)
    b. **Option 2:** Another video format uploaded directly.

2. **COLLABORATION:** Students can complete assignments in groups of 2 if desired. Both students should submit the work themselves however, unless they have a group in blackboard (email me to generate one).

GenAI Policy

GenAI chatbots may be used to aid in the assignment, but you cannot have them generate your code for you. Chatbots can be used to debug, explain concepts, point you to areas for improvement and write documentation. In-line editors such as copilot are allowed to speed up writing your code, but should not be prompted to complete the assignment on their own for you.