Senior Staff Software Engineer - Domain Challenge: Credit Reporting

Albert Einstein is often quoted as having said: "I know not with what weapons World War III will be fought, but World War IV will be fought with sticks and stones". We now know that World War III will be fought with memes. Big tech companies are now scrambling to find and deliver the latest memes to their users before their competitors can do so. Billions of dollars are spent in this arms race and companies are desperate to find easier ways to find memes.

This is where Esusu offers a solution. We have developed a new product called MaaS: Memes-as-a-Service. With MaaS, any company can access memes on demand via an API call, making it trivial to offer content to their customers on demand. For example, Instagram can make an API call to Esusu MaaS whenever a user opens their app and fetches a batch of memes to then show on the user's feed. MaaS even allows clients to customize memes based on queries and other metadata so they can tailor the content to their users' taste. For example, X/Twitter can ask for memes relevant to a user's current GPS location or based on key words for that user ("show me memes relevant to this Italian restaurant on a Friday date night").

Needless to say, MaaS has been a smash hit. Every name-brand tech company on the planet is lining up to access the Esusu memes API. As great as this sounds, there is one problem now – how do we keep up with this demand?

That's where **you** come in. You are being brought on to help scale the MaaS product. In this exercise, you will help optimize and operationalize our meme engine so we can continue to be the market leader in this competitive space.

Technical Overview

MaaS is an API product for fetching memes. At its core, it basically consists of one API call:

```
GET /memes/
```

This API call allows for parameters in the URL for metadata. For this exercise, you can assume that 3 types of metadata are allowed: latitude, longitude, and query (free text). So for example, a client could call:

```
GET /memes?lat=40.730610&lon=-73.935242&query=food
```

This should return a meme relevant to the query 'food' in the NYC area.

1. Create a microservice written in ***Golang*** which implements this API. The format of the response payload and how you generate a meme is up to you (points for creativity). For

example, you could randomly generate text-based memes, you could return a URL to an image meme from another site, or you could just return a dummy payload. The purpose of this exercise is to set up for the next question where we add features to the service.

2. We want to charge clients on a per-API-call basis. Specifically, we want to build a system where each client can purchase tokens which are then used to make API calls.
   a. Design and implement a system by which you can track the number of times a client has called the memes API. You can assume that the client will include an auth token in the request header which can be used to track their identity. Token balance should be stored in a database to keep track of usage.
   b. Design and implement a system by which you can produce in real-time a client's current token balance. You can assume that the process by which a client actually purchases tokens is handled by another team, and that this other team would make an API call to let you know when to add tokens to a client's balance.
   c. Data should persist when the server is restarted.
   d. Support 100 requests per second.

For question 2, you can implement a solution with the parameters listed above. We will scale it in the next question.

3. Explain in as much detail as you like how you would scale this API service to ultimately support a volume of 10,000 requests per second. Some things to consider include:
   a. How will you handle CI/CD in the context of a live service?
   b. How will you model and support SLAs? What should operational SLAs be for this service?
   c. How do you support geographically diverse clients? As you scale the system out horizontally, how do you continue to keep track of tokens without slowing down the system?

4. The success of this product has led to the team adding new and exciting features which the customers have highly requested. Specifically, we are now offering a premium offering: Memes AI. With Memes AI, you can get *even spicier* memes curated by generative AI. Naturally, this feature costs extra money and requires a separate subscription.

Describe how you would modify the service to now keep track of whether a client is *authorized* to get AI-generated memes. If a client has this subscription, then they should get AI-memes, and they should get normal memes otherwise. How do you keep track of authorization of a client as we cale the system without slowing down performance?