Exponea AI/Data Pipeline engineer hiring assignment

Task

Write an HTTP server. The server should serve 3 endpoints. All endpoints should perform a similar task, with a slight modification (described below). When handling an HTTP request, the endpoints should perform 3 HTTP requests to the Exponea testing HTTP server (specification below) and return the results to the caller. All endpoints are expected to accept a "timeout" parameter - integer value specifying time in milliseconds. Endpoints must respect this timeout value and always return the response within a given timeout.

You can choose to implement the assignment either in **Python** or **Go** language.

Endpoints - implemented by you:

1. **/api/all** - collects **all successful responses** from Exponea testing HTTP server and returns the result as an array. If timeout is reached before all requests finish, or none of the responses were successful, the endpoint should return an error.
2. **/api/first** - returns the **first successful response** that returns from Exponea testing HTTP server. If timeout is reached before any successful response was received, the endpoint should return an error.
3. **/api/within-timeout** - collects **all successful responses that return within a given timeout**. If a timeout is reached before any of the 3 requests finish, the server should return an empty array instead of an error. (This means that this endpoint should never return an error).
4. BONUS: **/api/smart** - instead of firing all 3 requests at once, this endpoint will first fire only a single request to Exponea testing HTTP server. Then 2 scenarios can happen:
    a. Received a successful response within *300 milliseconds*: return the response
    b. Didn't receive a response within *300 milliseconds*, or the response was not successful: fire another 2 requests to Exponea testing HTTP server. Return the **first successful response** from any of those 3 requests (including the first one).

A successful response is defined as a response, which returns HTTP status code 200, with valid payload.
Your endpoints should return a JSON response. By "returning the result as an array", we mean that you should combine successful responses from the Exponea testing HTTP server and return them as JSON array. Example responses from Exponea testing HTTP server: `{"time": 200}, {"time": 300}`; endpoint response: `[{"time": 200}, {"time": 300}]`.

Exponea testing HTTP server

The server serves a single endpoint, which performs some work (taking roughly 100-600ms), then returns the time it took to respond.

Endpoint:
GET `https://exponea-engineering-assignment.appspot.com/api/work`
Successful JSON response example: `{“time”: 160}`

Note - the server is not very reliable, which means that it doesn’t always return a successful response .

What we want to see

- Performance - it is expected that requests to Exponea HTTP server will be done concurrently. We will also test how your server implementation behaves under load - both in terms of requests per second, and the number of concurrent requests.
- Robust implementation - the server should behave correctly in presence of errors. Exponea testing HTTP server is infamously known for being unreliable. Also, the implementation should not leak resources (stuck threads/goroutines/connections for unknown time).
- Readable, maintainable implementation - for example, prefer well-known open source libraries, with licenses that allow them to be used commercially.

Extras - if you want to shine

- Write tests for your implementation
- Instrument your code with tools, that will make debugging in production easier - logging, monitoring, tracing. You want to have a way to find out if Exponea testing HTTP server is behaving well in production.
- Include Dockerfile that can be used to build and run the server.
- Attach discussion about code behavior - known edge cases, how it behaves in certain conditions, performance characteristics, resource requirements, etc. How many concurrent requests can the server handle? How would you protect the server against being overloaded?