Interview Coding Assignment

# Instructions

The following is an assignment which is used to help evaluate potential candidates. It is expected that this assignment should take approximately 2 hours. Candidates should complete the assignment within 2 weeks of receiving the instructions.

Using generative AI technologies is encouraged and you are free to use any generative AI technology that you prefer. While AI should be used as an assistant, you will be responsible for any and all of the content of your submission.

While this is just a sample application, you should approach it in the same way you would in a professional environment. Others will review your code, show your work and process.

During one of the technical interviews, you will be expected to:

* Explain any of the code that was submitted and any reasoning behind it.
* How you used generative AI technology and what approaches you used.
* Make a specific change to the code in a live coding session. This session will not allow the use of AI tools. You will be allowed to use any IDE of your choice and will need to share your screen during this session.

Once the assignment is completed, please upload it as a private Github repository that is shared with the following Github accounts: [mwringe](https://github.com/mwringe), [pavolloffay](https://github.com/pavolloffay), and [frzifus](https://github.com/frzifus).

You are only given access to the information in this document. Please include an **Assumptions.md** file to include any assumptions you made about the implementation.

# 

# Assignment

**Scenario**: We are building a high-throughput storage middleware. We need a service that acts as a Content-Addressable Storage (CAS) layer. The goal is to minimize disk footprint by ensuring identical content is never stored twice, regardless of which bucket or path it belongs to.

**Task**: Implement a HTTP service in Go that stores objects organized by buckets with de-duplication.

## Core Requirements:

1. **Strict Deduplication:** If bucket A/object 1 and bucket B/object 2 have the exact same content, the binary data must be stored only once on the underlying storage.
2. **Concurrency:** The service will be hit by concurrent requests. Ensure storage logic is thread-safe (e.g., avoid race conditions where a blob is deleted by User A just as User B is trying to read it).
3. **Production Standards:**
   * **Performance:** The service should be able to handle a high number of concurrent requests.
   * **Configuration:** All parameters (port, storage path, etc.) must be configurable via Environment Variables.

## 

## API Specification:

* **PUT /objects/{bucket}/{objectID}**
  + Reads the request body. Calculates the hash.
  + If the hash exists, increment the reference count and link the ID.
  + If the hash does not exist, store the blob, set ref count to 1, and link the ID.
  + *Returns:* 201 Created
* **GET /objects/{bucket}/{objectID}**
  + *Returns:* 200 OK (The content), or 404 Not Found.
* **DELETE /objects/{bucket}/{objectID}**
  + Decrements the reference count for the underlying hash.
  + If ref count == 0, delete the blob from storage.
  + *Returns:* 200 OK or 404 Not Found.

## 

## Deliverables:

1. **Go Source Code:** Organized cleanly (e.g., separate handler, storage, and domain layers).
2. **Testing:** unit and integration tests.
3. **Assumptions.md:** Documented assumptions about implementation.