JumpCloud Interview Assignment

Prerequisites

- You will need a computer that meets the system requirements of running the Go tools.
- Basic Git knowledge
- A GitHub.com account

Setup

Install Go Tools

If you are new to Go please follow the install instructions to get the Go tools installed and working on your machine. After you have completed the “Test Your Installation” section you should be ready to proceed.

Directory Structure

By default Go assumes that your `$GOPATH` is set to `$HOME/go`. If you use the defaults, your environment will require no additional setup after you have installed the Go Tools.

Create the directory
`$HOME/go/src/github.com/<your_github_username>/<name_of_your_repo_for_the_assignment>`
In Go, all files in a directory have to be in the same package. If you plan to use more than one package you should create sub-directories.

You should now be ready to start the assignment.

Requirements

The project code should only import packages available in the standard library.

Please complete the following requirements in order. Depending on how much time you have you may not get to the end. By completing the requirements in order you can always remain at a good stopping point. There are no super secret hidden solutions. The assignment is designed to see how you approach writing code to solve problems. We are also looking to see code that represents your view of “production quality”. If at any step along the way you have questions feel free to reach out.

Note: Please do not submit separate solutions for each of the steps. The assignment is designed to be cumulative so it is not necessary to, for example, to provide a separate file or branch for each step. Instead just let us know which step you finished when submitting your assignment.

To submit your work please push to a public GitHub project, and be sure to document any configuration or run instructions. We're looking for a solution to the problem as well as attention to detail and code craftsmanship. Good luck and have fun!

1. Hash and Encode a Password String

Provide the ability to take a password as a string and return a Base64 encoded string of the password that has been hashed with SHA512 as the hashing algorithm.

For example, if given the string `"angryMonkey"` the expected return value is `"ZEHhWB65gUlzdVwtDQArEyx+KVLzp/aTaRaPlBzYRIFj6vjFdqEb0Q5B8zVKCZ0vKbZPZklJz0Fd7su2A+gf7Q=="`

2. Hash and Encode Passwords over HTTP

Change your program so that when launched your code starts and listens for HTTP requests on a provided port. Accept `POST` requests on the `/hash` endpoint with a form field named `password` to provide the value to hash. The response should be the base64 encoded string of the SHA512 hash of the provided password. The server should not respond immediately, it should leave the socket open for 5 seconds before responding.

For example: `curl –data "password=angryMonkey" http://localhost:8080/hash` should return `"ZEHhWB65gUlzdVwtDQArEyx+KVLzp/aTaRaPlBzYRIFj6vjFdqEb0Q5B8zVKCZ0vKbZPZklJz0Fd7su2A+gf7Q=="` after approximately 5 seconds.

Note: Your software should be able to process multiple connections simultaneously.

3. Graceful Shutdown

Provide support for a "graceful shutdown request". If a request is made to `/shutdown` the server should reject new requests. The program should wait for any pending/in-flight work to finish before exiting.

4. Statistics Endpoint

Provide a statistics endpoint to get basic information about your password hashes.

A `GET` request to `/stats` should return a JSON object with 2 key/value pairs. The `"total"` key should have a value for the count of password hash requests made to the server so far. The `"average"` key should have a value for the average time it has taken to process all of those requests in microseconds.

For example: `curl http://localhost:8080/stats` should return something like: `{"total": 1, "average": 123}`