Hi team,

Below are some notes to help support my response for part1 of the coding task. Thanks in advance for reading:)

Walkthrough of how I arrived at this solution:

1. **Read and understand the basic requirements**: this seems to be a simple project with a typical client-server interaction between the service-to-build and the API handler(which could be either public or self-implemented, as stated in the assignment)
2. **Determine the structure and architecture of the Go application**: since the aim is to only have a running service that retrieves input, validates it and sends it to another API, I think it's sufficient and suitable to build a Go application that is as simple as possible, which means:
    a. I didn't write any unit tests or integration testing
    b. I didn't employ the classic "handler - usecase - repository" service architecture to separate the responsibilities, instead I put everything inside main.go as a simple and compact program.
    c. I didn't use a separate .env file to specify the variables, instead I hardcoded variables such as API-URL into the program.
    Again, in real world Go applications where things like db connection or config for other micro-services is needed, I would use a .env file to separate these config-related variables.
3. **Determine the delivering of the Go application:** I went for the simplest solution, since it was mentioned that I could even just provide a Go playground link haha.
   But I am aware of the potential add-ons to this project when it comes to real-world application delivering (which is also what I always did back in my previous job):
    a. Use a docker container to unify the runtime environment for microservices is a standard practice.
    b. Implement CI/CD pipelines to automate builds, tests and deployment of the services.
4. **Decide what to use for the API**: I built another simple service and run it as a separate service for handling the request. (and of course, alternatively, I could choose to search for a free and valid API on the internet insteading of having to run 2 services)

Solution Overview:

Overall there are **two Go applications**, one sitting under cmd/client which is responsible for reading input from JSON, validating the input and sending it to an API; the other one sitting under cmd/server is the one that acts as the API.
I also extract the common struct definition for the input data format into /pkg/common.go

Challenges, Considerations and Requirements:

I think for me the main challenge throughout part1 was to carefully balance the simplicity and complexity of the application, since I wanted to deliver a solution that is effective and suitable for the scope of this task, while also being able to demonstrate my understanding and knowledge of the programming language and industrial-level practice of Golang/backend development.
As for considerations and requirements, I think I have addressed them in the above sections.

Thank you so much for reading, and if there’s anything you’d like me to demonstrate or discuss, please reach out to me and I’m more than happy to do that!