Overview

This solution reads JSON in from a file, validates it against basic rules and forwards the input as part of a post request to an HTTP listener in this order.

Requirements

Retrieve JSON input.

Validate JSON input.

Log input to STDOUT stream.

Send request to HTTP API with the input.

Considerations

I assumed the input was coming from a file.

When thinking about my solution I considered whether operations can be done in parallel, for this problem I did not find any so everything is performed synchronously.

I split my functions up in such a way that it wouldn't be a big ask for someone to extend the program to take multiple inputs and send multiple requests granted the input is of the same format.

Walkthrough of my thought process writing the code

Firstly I read the full requirements of the task to gauge whether the solution could be done using goLangs concurrency tools however the operations of reading, validating and making a request all need to come in sequential order.

I mapped out my expected JSON input to a struct where all fields are tagged as JSON so I could use the encoding/json package to unmarshal into my struct.

Following this I wrote the readInput function which takes a filepath, reads the contents out and unmarshals it into my inputStructure struct and returns a pointer to it granted there are no errors. I chose to declare my struct inside this function and return a pointer to it rather than declaring it in main and passing the pointer into the function in case this program needs to be extended, having the function generate the struct means you can call the function every time you need one rather than only getting one per program run.

I then did the validation using the validator package I found on github. It lets you tag your struct fields with validation rules. I picked some basic ones to demonstrate it.

Now I have all my data processed I can form my request and send it to an endpoint. I remarshal my struct back to a byte array to send with my request, often API endpoints have some sort of auth so I went ahead and added basic authentication as a header and send it off.

Other

I setup an http listener for index on localhost port 8080 as part of the program to show it working. Since LitenAndServe is blocking I had to send it off to its own goroutine, if listening was part of this programs requirements I would use some sort of synchronization to confirm that the socket was ready to receive requests before going ahead and sending it but its not a big deal here.

Although I do not have to unpack the JSON into a struct to validate it, having the data in a struct is incredibly useful for any required manipulations of the data along with making it easy for other developers to pick it up and extend/modify. I think the extra little bit of memory is worth it.