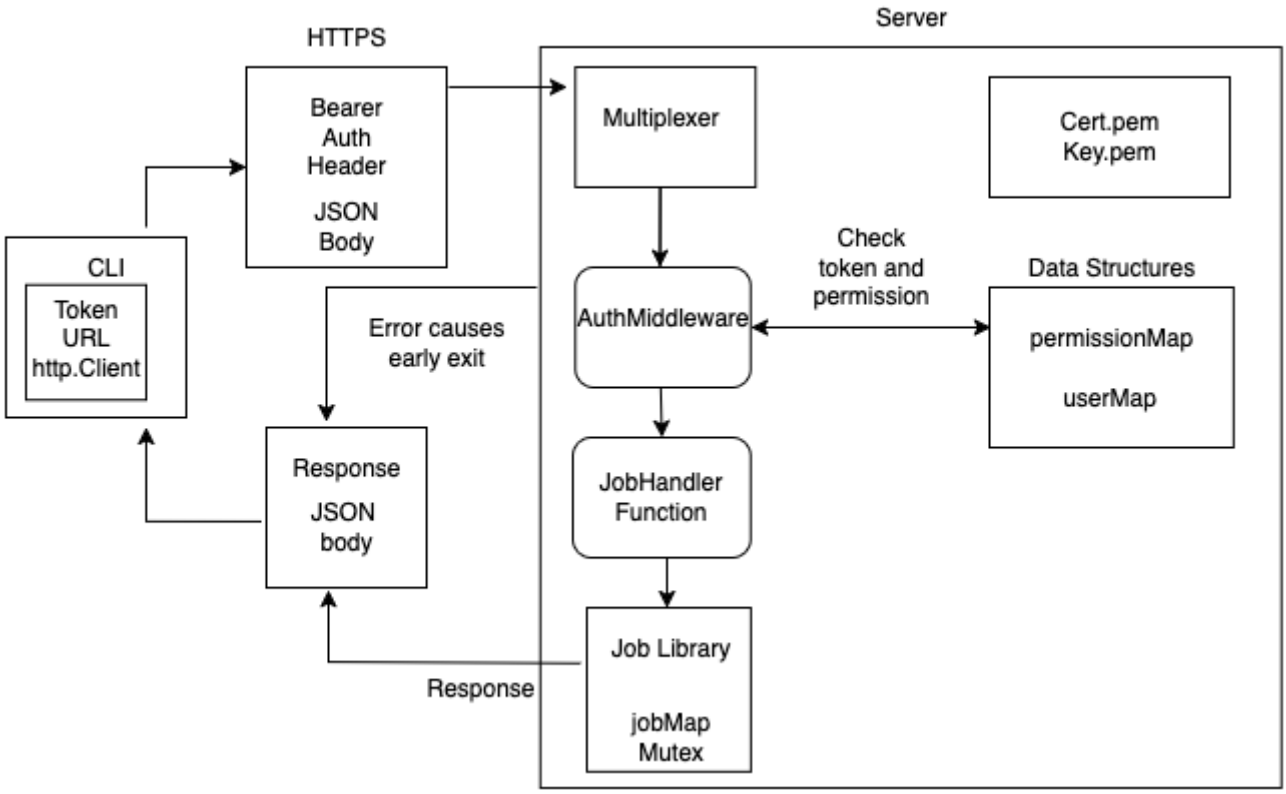
JobWorker Design Doc



Proposed API

- Four endpoints: `/jobs/start`, `/jobs/stop/{id}`, `/jobs/status/{id}`, `/jobs/output/{id}`
- If the server is running on `https://localHost:8443`, the endpoint for start would be `https://localHost:8443/jobs/start`
- `"/jobs/"` prefix included to delineate this API from other possible endpoints, e.g. `/users/`
- `start` and `stop` are `POST` methods, `status` and `output` are `GET` methods
- Expected status codes: 404 (resource does not exist) 400 (does not have ID) 201 (job started) 200 (request fulfilled) 401 (authentication failed) 403 (authorization failed)
- Response bodies will be made for each job type rather than sending the whole job struct to client, start is the only request that will need a body since the ID is in the URL
  - Request: `POST /jobs/start Auth Bearer "admin-846a7" {"cmd": "echo, "args": ["Hello World"]}`
  - Response: `201 {"job_id": "4c20a846-a7...",}`

Component Details

1. **Worker Library**
   - Data Structures:
     - `Job` struct with fields like ID, status, stdout, stderr, cancel function
     - `jobMap`, `jobLock`: Mapping from `ID` to `Job`, and a read/write lock `sync.RWMutex`
   - Functions to perform tasks from the API (all public):
     - `Start() (id string, err)`, `Stop(id string) err`, `Status(id string) (string, err)`, `Output(id string) ([]byte, string, err)`

- Notes:
  - Will use Google's UUID package for generating job IDs
  - When a command is started via `Start()` it will be given a context with cancel function for termination in the `Stop()` function
  - To stop the job, the stop function will look up the `Job` structure using its `ID` and call its `cancel()` function
  - Will lock `jobMap` appropriately to prevent data races but allow concurrent reads

2. **Server**
   - Data Structures:
     - `User` struct with fields for `TokenID string` and `Role string`
     - `userMap map[string]User` map of users where key is `TokenID`
     - `rolePermissions map[string]map[string]bool` a map to authorize api actions based on `User.role`. First key is role, second key is action.
     - `mux` a multiplexer to route API using `http.NewServeMux()`
   - Functions:
     - `handleStartJob(w http.ResponseWriter, r *http.Request)`, `handleStopJob()`, `handleGetJobStatus()`, `handleGetJobOutput()`: helper functions for executing job tasks
     - `authMiddleware(action string)` Authenticates user and authorizes provided action
   - Auth Workflow Overview:
     - TLS handshake -> client makes request to server with Bearer Token -> server multiplexer directs route -> authenticate -> authorize action -> execute command -> response to client.
   - Authentication:
     - Use Bearer Token authentication by checking the HTTP header's auth field for a token and finding the `User` in `userMap` using the token string as a key.
   - Authorization:
     - The server will maintain a `rolePermissions` map in the format `{role : {action : isAllowed}}` for task authorization based on the user's role.
     - Each user is assigned a role. To check whether a user can perform an action, the server looks up their role in the outer map and then checks the inner map for that action
     - The inner map acts as a set
   - Auth Middleware:
     - The authMiddleware function will be called with the appropriate `action` to validate if user can perform a requested api action.
   - HTTPS:
     - Generate and hard code self signed certificate, client skips verification
   - Notes:
     - Tokens for 3 user roles will be premade for demo: `admin`, `user`, `viewer`
3. **Client**
   - Data Structures:
     - `Client` struct containing `token`, `baseURL`, and `http.Client`
   - Functions: Most functions will act on the `Client` struct as a method
     - `MakeClient` called in main with parsed command line flags `-token` and `-server`, creates new `Client`

        - `(c *Client) makeRequest()` helper function that builds and sends a request to the server, called within following helpers
            - `StartJob(command string, args []string) (string, error)` starts a job with given parameters
            - `StopJob(id string)` stops the job with id
            - `GetStatus(id string)` gets status of job with id
            - `GetOutput(id string)` gets output of job with id
    - Workflow: (Assumes server is running)
        - `main()` makes a new client, sets client variables from command line or defaults to `user` token and default server path `https://localhost:8443`
        - Calls appropriate helper for the inputted action, which sends request to server and gets a response
        - Return a confirmation, error, or response to user
    - Notes:
        - Decided to make a client type in order to have the user token and server URL be alterable from CLI using flags

Security Considerations

- Uses Go (version 1.23.2) default TLS (min 1.2) as well as Go's default cipher suite ordering
- Uses self signed certificates, and client bypasses verification (if server is public, potential for MITM)
- Bearer tokens are stored unencrypted in memory, they never expire or rotate
- User can run any linux command their privilege level allows means potential for malicious commands
- No input validation means a potential for injection attacks
- No sandbox or containerization on the server, client can access server's file system
- No job timeout or job resource limitation (I considered role based default timeout for jobs)

CLI UX

Will likely be modified during development to be more user friendly (better spacing, etc). Sample assumes that the server is running using the default token and server URL values

```
$~jobWorker start echo "Hello World"
Job started with ID 4c20a846-a780-4378-88bc-2447eb072811
$~jobWorker start sleep 30
Job started with ID bf779987-12cf-41a8-b991-21045cd5d822
$~jobWorker stop bf779987-12cf-41a8-b991-21045cd5d822
Job stopped with ID bf779987-12cf-41a8-b991-21045cd5d822
$~jobWorker output 4c20a846-a780-4378-88bc-2447eb072811
Hello World
$~jobWorker status 4c20a846-a780-4378-88bc-2447eb072811
Succeeded
$~jobWorker stop bf779987-12cf-41a8-b991-21045cd5d822
Error: Job is not running
```