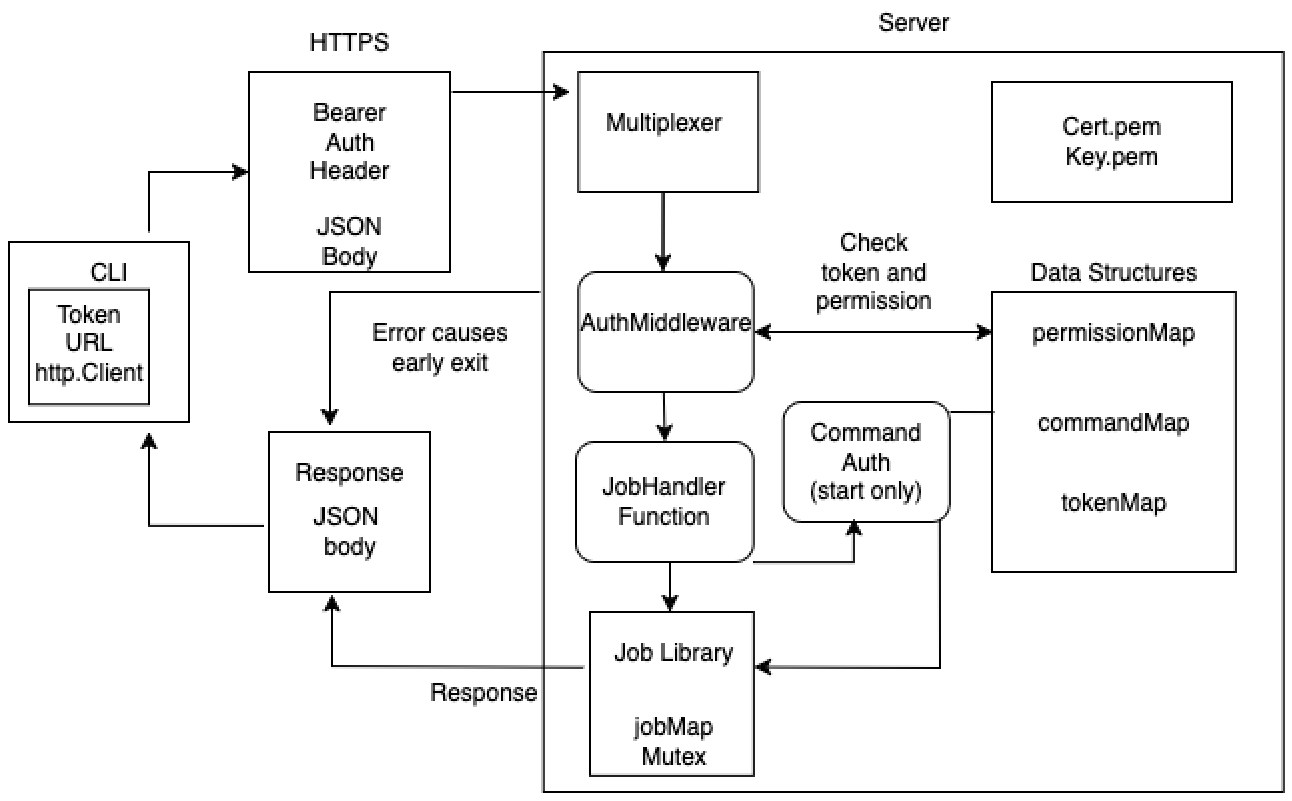
JobWorker Design Doc



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/`

Component Details

1. **Worker Library**
   - Data Structures:
     - `Job` struct with fields like ID, status, output, cancel
     - `jobMap`, `jobLock`: Mapping from `ID` to `Job`, and a read/write lock `sync.RWMutex`
   - Functions to perform tasks from the API (all public):
     - `Start()`, `Stop()`, `Status()`, `Output()`
   - Notes:
     - Will use Google's UUID package for generating job ID's
     - When a command is started via `Start()` it will be given a context with cancel for termination in the `Stop()` function
2. **Server**
   - Data Structures:
     - `Token` struct with fields for `TokenID string` and `Role string`
     - `tokenMap map[string]Token` a map of `Token` structs that are set to default values

    - `rolePermissions map[string]map[string]bool` a map to authorize api actions based on `Token.role`
    - `commandPermissions map[string][]string` maps `role` to a list of linux commands that role can run
    - `mux` a multiplexer to route API using `http.NewServeMux()`
  - Functions:
    - `handleStartJob(w http.ResponseWriter, r *http.Request)`, `handleStopJob()`, `handleGetJobStatus()`, `handleGetJobOutput()`: helper functions for handling parsing and executing api routes
    - `authenticate(r *http.Request) bool` Authenticate function which extracts the token and checks if it is in the `tokenMap`
    - `authorizeAction(role, action string) bool` Checks if a user can perform an api action (start, stop)
    - `authorizeCommand(role, command string) bool` Checks if a linux command is allowed for a given role
    - `authMiddleware(action string)` Authenticates user and authorizes action (start, stop, output)
  - Auth Workflow Overview:
    - TLS handshake -> client makes request to server with Bearer Token -> server multiplexer directs route -> authenticate -> authorize action (start, stop) -> authorize linux command (job starting only) -> execute command -> response to client.
  - Authentication:
    - Use Bearer Token authentication by checking the HTTP header's auth field for a token and finding the `Token` in `tokenMap`
  - Authorization:
    - Two separate authorizations: api action (start, stop, status, output) and command (ls, grep, rm)
    - The server will maintain a `rolePermissions` map in the format `{Permission Level : {job : isAllowed}}` for task authorization
    - The linux command authorization will only be checked if a job is starting and will check the `commandPermissions` map
  - 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:
    - Decided to include both role based authz (viewer cannot stop job or start job) as well as command based authz (regular user cannot run `rm` but can run `ls`) for a small amount of added security. Will omit one if requested in design review
    - Tokens for 3 user roles will be premade for demo: `admin`, `user`, `viewer`
    - Jobs stored in memory, no persistance

3. **Client**
  - Data Structures:
    - `Client` type containing `token`, `baseURL`, and `http.Client`
  - Functions: Most functions will act on the `Client` type 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 flagged variables or defaults to admin token and default server path `https://localhost:8443` (for demo)
        - Parses args
        - Uses a switch statement for `args[0]`: start, stop, status, output
        - Calls appropriate helper, which sends request to server and returns a response
        - Return a response, error, or confirmation to user
    - Notes:
        - Decided to make a client type in order to have the user token and server URL be alterable from CLI

Security Considerations

- Uses self signed certificates, and client bypasses verification
- 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 but thought it was too much)

CLI UX

Will likely be modified during development to be more user friendly (better spacing, etc)

```
            $~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
```