Tesi Laboratorio di amministrazione dei Sistemi MeShell

Davide Ferrara 518629

November 7, 2025

Abstract

MeShell è una piattaforma per l'amministrazione remota di sistemi basata sul web. Esse fornisce uno o piú terminali SSH accessibili tramite browser, arricchito da una serie di comandi rapidi pensati per semplificare le operazioni comuni di un amministratore di sistema. Questa relazione analizza in modo approfondito l'architettura di MeShell, dissezionando il codice sorgente del backend in Go, del frontend in JavaScript e dello script meshell.sh per fornire una comprensione completa del suo funzionamento interno, con un focus specifico sulle pratiche di amministrazione di sistema.

Contents

1 Architettura Generale

MeShell segue un'architettura client-server a tre livelli:

1. **Frontend Web:** Un'interfaccia utente (UI) reattiva che l'amministratore di sistema utilizza tramite un qualsiasi browser moderno.
2. **Backend Server:** Un server monolitico scritto in Go che funge da cuore del sistema. Ha il duplice ruolo di web server per il frontend e di gateway WebSocket per la comunicazione in tempo reale. La sua responsabilità principale è orchestrare le sessioni SSH verso le macchine remote.
3. **Meshell.sh :** Su ogni macchina risiede lo script `meshell.sh`, un componente chiave che astrae comandi complessi in scorciatoie semplici.

1.1 Installazione

Per utilizzare Meshell é necessario innanzi tutto clonare il repository tramite:

```
git clone https://github.com/davide-ferrara/meshell.git
cd meshell
```

Successivamente é necessario intallare le dipendenze:

```
go mod tidy
npm install
```

Fatto ció per avviare il Server Web basterá eseguire:

```
npm run dev
```

All'interno dei nostri client invece basterá installare lo script Meshell eseguendo:

```
chmod +x install.sh
./install.sh
```

Il seguente script creerá un link simbolico in /usr/bin che permetterá si eseguite i vari comandi Meshell all'interno dei client.

```
origin="$(pwd)/meshell.sh"
dest="/usr/local/bin/meshell"

sudo ln -sf "$origin" "$dest"
for f in bin/*; do
sudo ln -sf "$(pwd)/$f" "/usr/local/bin/$(basename "$f")"
done
```

Sará possibile visualizzare Meshell all'indirizzo "**http://localhost:5173/**".
La directory di installazione di default di Meshell é **/usr/share/meshell**.

1.2 Gestione dei permessi

La gestione dei permessi é un punto cruciale dell'applicazione. Mettiamo caso che Meshell venga utilizzato all'interno di un laboratorio universitario composto da diverse macchine Linux e che lo script Meshell sia installato in ognuna di queste. Supponiamo anche che ogni macchina sia caratterizzata da almeno due gruppi: "**Admin**" e "**Studenti**". É importante assicurarsi che il proprietario (**owner**) della cartella *"usr/share/meshell/"* sia un utente appartente al gruppo "Admin" e non "Studenti" in modo tale che gli script non possano essere modificati da utenti diversi dall'amministratore di sistema.

É possibile fare ció utilizzando i comandi:

```
# Visualizza i permessi
ls -l /usr/share/meshell
# Nel caso sia proprietario il gruppo Studenti
chown -R admin1:admin /usr/share/meshell
```

1.3 Backend (server.go)

Il server, scritto in Go, è il componente centrale che permette di visualizzare via web tramite protocolli (HTTP/WebSocket) un terminal (SSH).

1.4 Avvio e Routing

Il punto di ingresso dell'applicazione è la funzione `main`, che inizializza il server HTTP e imposta il routing.

```
func main() {
    flag.Parse()
    log.SetFlags(log.Ldate | log.Ltime | log.Lshortfile)

    log.Println("Meshell Server is starting...")

    http.HandleFunc("/tty", tty)
    http.HandleFunc("/", home)
    http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(http.Dir("
    static"))))

    log.Printf("Server listening on address %s", *addr)
    err := http.ListenAndServe(*addr, nil)
    if err != nil {
        log.Fatalf("Fatal error: %v", err)
    }
}
```

Oltre a servire i file statici (CSS, immagini), il router definisce due handler principali:

- `home`: Gestisce la rotta radice `/` e serve il file `index.html`.
- `tty`: Gestisce la rotta `/tty`, questo é l'endpoint che effettua l'upgrade della connessione da HTTP a WebSocket e permette a Xterm.js il flusso dei dati dal pty.

1.5 La Sessione Terminale (tty handler)

Quando il frontend invia una richiesta all'endpoint `/tty`, l'handler esegue una sequenza di operazioni critiche che stabilisce la sessione interattiva, ovvero un flusso bidirezionale di byte interpretati successivamente in caratteri. In seguito le principali operazioni che svolge il server:

1. **Apertura WebSocket:** Utilizzando la libreria `gorilla/websocket` la connessione HTTP iniziale viene trasformata in una connessione WebSocket bidirezionale.
2. **Connessione SSH:** Invoca la funzione `connectSSH` per stabilire una connessione SSH con le macchine remote, utilizzando credenziali pre-configurate.
3. **Setup della Sessione PTY:** Chiama `setupSSHSession` per richiedere uno pseudo-terminale (PTY) e avviare una shell remota.
4. **Avvio del Piping:** Infine, `startPiping` avvia le goroutine necessarie per inoltrare i dati in entrambe le direzioni: dal WebSocket allo stdin della shell SSH e dallo stdout/stderr della shell SSH di nuovo al WebSocket.

In Linux la maggior parte degli **Emulatori di Terminale** (come gnome-terminal o alacritty) utilizzano una shell come **Bash** per eseguire comandi e script. Esistono poi anche altre varianti di shell, come ad esempio Fish.
Bash non è altro che un programma, che è possibile eseguire da **/bin/bash**, ma come fa l'Emulatore di Terminale a comunicare con esso?

Dopo l'avvio, l'Emulatore di Terminale richiede al kernel (aprendo /dev/ptmx) un nuovo pseudoterminale. Il kernel gli restituisce un file descriptor (l'handler) per il **PTM (Pseudo Terminal Master)**, che farà da "pipe" verso l'altro capo, il **PTS (Pseudo Terminal Slave)**.

Il PTS è un file contenuto in /dev/pts/ (es. /dev/pts/5). Questo file verrà usato dalla shell (bash) che l'emulatore avvia trmaite **fork()** per creare un processo figlio.

Questo processo figlio, prima di trasformarsi in bash (tramite exec()), imposta i suoi **STDIN**, **STDOUT** e **STDERR** al file **PTS** (es. /dev/pts/5) ed in questo modo il processo padre "Terminale" e il processo figlio "Bash" avranno stabilito una comunicazione bidirezionale detta anche **IPC**.

```
~ > pstree
systemd
|             ...
|
|-alacritty---bash---pstree
|
|
```

Da questo output del comando pstree possiamo vedere come il Terminal Emulator, abbia come figli Bash e pstree che é il comando eseguito. La logica di Meshell é proprio questa, il nostro frontend é il Terminal Emulator (un ambiente grafico), il Server Go non é altro che il ponte (come il kernel) e il client SSH il PTY.

1.6 Connessione SSH (connectSSH)

Questa funzione astrae la logica di connessione SSH.

```go
func connectSSH(user string, password string, addr string) (*ssh.Client, error) {
    // ... (gestione home dir)
    khPath := filepath.Join(home, ".ssh", "known_hosts")
    hostKeyCallback, err := knownhosts.New(khPath)
    if err != nil {
       // Host Sconosciuto
    }

    config := &ssh.ClientConfig{
        User: user,
        Auth: []ssh.AuthMethod{
            ssh.Password(password),
        },
        HostKeyCallback: hostKeyCallback,
    }

    return ssh.Dial("tcp", addr, config)
}
```

Dal punto di vista della sicurezza, è importante notare il meccanismo di `HostKeyCallback`. Il codice tenta di utilizzare il file `.ssh/known_hosts` per verificare l'identità del server remoto, proteggendo da attacchi man-in-the-middle.

1.7 Creazione del PTY (setupSSHSession)

Questa è la funzione dove viene creato l'ambiente terminale interattivo.

```go
func setupSSHSession(client *ssh.Client, rows int, cols int) (*ssh.Session, io.
    WriteCloser, io.Reader, io.Reader, error) {
    session, err := client.NewSession()
    // ...
    sshStdin, _ := session.StdinPipe()
    sshStdout, _ := session.StdoutPipe()
    sshStderr, _ := session.StderrPipe()

    modes := ssh.TerminalModes{
        ssh.ECHO:          1, // Abilita l'eco dei caratteri
        ssh.TTY_OP_ISPEED: 14400, // Velocita di input
        ssh.TTY_OP_OSPEED: 14400, // Velocita di output
    }

    if err := session.RequestPty("xterm", rows, cols, modes); err != nil {
        // ...
    }

    if err := session.Shell(); err != nil {
```

```
        // ...
    }

    return session, sshStdin, sshStdout, sshStderr, nil
}
```

La chiamata a `session.RequestPty` è fondamentale. Chiede al server SSH di allocare uno pseudo-terminale, specificando il tipo di emulazione (`"xterm"`) e le dimensioni. Senza un PTY, non sarebbe possibile avere una shell interattiva, ma solo eseguire comandi singoli. Subito dopo, `session.Shell()` avvia la shell remota associata a questo PTY.

1.8 Flusso Dati Bidirezionale (startPiping)

Questa funzione orchestra il flusso di dati tra il client web e la shell remota tramite tre goroutine concorrenti.

```
func startPiping(ws *websocket.Conn, sshStdin io.WriteCloser, sshStdout, sshStderr
    io.Reader, sshSession *ssh.Session) error {
    var wg sync.WaitGroup
    wg.Add(3)

    // Goroutine 1: WebSocket -> SSH Stdin
    go func() {
        // ... (legge dal ws e scrive su sshStdin)
    }()

    // Goroutine 2: SSH Stdout -> WebSocket
    writer := &websocketWriter{ws: ws}
    go func() { defer wg.Done(); copyStream(writer, sshStdout) }()

    // Goroutine 3: SSH Stderr -> WebSocket
    go func() { defer wg.Done(); copyStream(writer, sshStderr) }()

    err := sshSession.Wait()
    wg.Wait()
    return err
}
```

La prima goroutine legge i messaggi dal WebSocket (input dell'utente) e li scrive nello `stdin` della sessione SSH. Gestisce anche i messaggi speciali di "resize" per ridimensionare il PTY. Le altre due goroutine leggono continuamente da `stdout` e `stderr` della sessione SSH e scrivono ogni dato ricevuto nel WebSocket, inviandolo così al browser dell'utente.

1.9 Frontend (index.js)

Il frontend si occupa di presentare l'interfaccia all'utente e di gestire la comunicazione con il backend.

Figure 1: Interfaccia Web di MeShell

1.10 La Classe `Meshell`

Il codice è organizzato attorno a una classe `Meshell`, che incapsula la logica per un singolo terminale.

```
class Meshell {
  constructor(tname) {
    this.tname = tname;
    this.ws = null;
    this.fitAddon = new FitAddon();

    this.term = new Terminal({
      cursorBlink: true,
      fontSize: 18,
      // ... altre opzioni
    });
    this.term.open(document.getElementById(this.tname));
    this.term.loadAddon(this.fitAddon);

    // ... (gestione eventi)
  }
  // ... metodi
}
```

Il costruttore inizializza un'istanza della libreria `xterm.js`, la configura con un tema e un font, e la collega a un elemento `<div>` nell'HTML. Carica anche l'addon `FitAddon`, essenziale per far sì che il terminale si adatti alle dimensioni della finestra.

1.11 Connessione e Comunicazione

Il metodo `openConnection` è responsabile di stabilire la connessione WebSocket.

```
  openConnection() {
    // ...
    const socketURL = `ws://${location.host}/tty?rows=${rows}&cols=${cols}`;
    this.ws = new WebSocket(socketURL);
```

```

    this.ws.onopen = () => {
      this.term.focus();
      this.term.onData((data) => {
        if (this.ws && this.ws.readyState === WebSocket.OPEN) {
          this.ws.send(data);
        }
      });
    };

    this.ws.onmessage = (event) => {
      this.term.write(new Uint8Array(event.data));
    };
    // ...
  }
```

ahhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
Quando la connessione è aperta (`onopen`), il terminale riceve il focus e viene attivato l'handler `onData`. Da questo momento, ogni tasto premuto dall'utente viene inviato al server. Viceversa, ogni messaggio ricevuto dal server (`onmessage`) viene scritto direttamente nel terminale con `term.write()`, che si occupa di interpretare i dati e visualizzarli.

1.12 Gestione dei Comandi Rapidi

La funzione `setupCommandButtons` collega i pulsanti HTML all'invio di comandi.

```
function setupCommandButtons() {
  const commandButtons = document.querySelectorAll(".button-panel > button");

  commandButtons.forEach((button) => {
    button.addEventListener("click", () => {
      const command = button.id;
      // ...
      activeTerminal.sendCommand(`source meshell --cmd ${command}`);
      // ...
    });
  });
}
```

Per ogni pulsante, viene aggiunto un listener che, al click, recupera l'ID del pulsante (es. `"ps"`, `"update"`) e lo usa per costruire la stringa di comando `source meshell -cmd <id>`. Questa stringa viene poi inviata al terminale attivo tramite il metodo `sendCommand`.

2 Meshell.sh

Questo script Bash è un componente fondamentale che agisce come un dispatcher di comandi sulla macchina remota. Viene invocato con `source` per garantire che comandi come `cd` modifichino la shell corrente.

2.1 Struttura Principale

Lo script utilizza un costrutto `case` per interpretare il secondo argomento (`$2`) passato dopo l'opzione `-cmd`.

```
case "$1" in
  "--cmd")
    case $2 in
      "ps")
        # ... codice per ps ...
        ;;
      "update")
        # ... codice per update ...
        ;;
      # ... tutti gli altri comandi
    esac
```

```
    ;;
  # ... altre opzioni come start, stop, etc.
esac
```

2.2 Comandi dall'Interfaccia Web (Amministrazione Guest)

2.2.1 Controlli Rapidi

Figure 2: Interfaccia dei Controlli Rapidi

Clear Terminal Pulisce la schermata del terminale corrente. È un comando standard della shell.

List (ls) Elenca i file e le directory nella posizione corrente. Lo script `meshell.sh` usa l'alias `ls -la` per fornire un output dettagliato.

```
"ls")
    ls -la
    ;;
```

Navigazione (cd, pwd) I comandi `cd` (Change Directory) e `pwd` (Print Working Directory) sono gestiti per permettere una navigazione di base. I pulsanti rapidi offrono scorciatoie per le directory `..` (parente), `~` (home) e `/` (root).

```
"cd")
    echo -n "Inserisci il percorso: "
    read path
    cd $path
    ;;
"pwd")
    pwd
    ;;
```

Cambia Utente (su) Il comando `su -` permette di cambiare l'utente corrente, richiedendo la password dell'utente di destinazione. È un comando standard di sistema.

Zoom (+/-) Questi controlli sono una funzionalità del frontend e non vengono inviati al server. Permettono di aumentare o diminuire la dimensione del font nel terminale per migliorare la leggibilità, agendo direttamente sulla configurazione di Xterm.js.

Meshell Update Questo comando aggiorna lo script `meshell.sh` stesso, eseguendo un `git pull` dalla directory in cui è stato clonato il repository.

```
"meshell-update")
    # ... (codice per l'aggiornamento)
    ;;
```

2.2.2 Gestione Pacchetti (Basati su APT)

update Questo comando esegue l'aggiornamento completo del sistema, sincronizzando prima la lista dei pacchetti e poi installando le nuove versioni disponibili.

```
"update")
    echo "Eseguendo 'sudo apt update && sudo apt upgrade'..."
    sudo apt update && sudo apt -y upgrade
    ;;

```

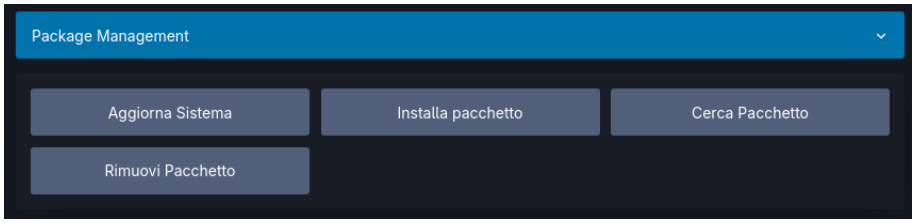

Figure 3: Interfaccia di Gestione Pacchetti

install Permette all'amministratore di installare un nuovo pacchetto. Il comando è interattivo e richiede di inserire il nome del pacchetto da installare.

```
"install")
    echo -n "Inserisci il nome del pacchetto da installare: "
    read package
    echo "Eseguendo 'sudo apt install $package'..."
    sudo apt install -y $package
    ;;
```

search Cerca un pacchetto nei repository configurati, utile per trovare il nome corretto di un software prima dell'installazione.

```
"search")
    echo -n "Inserisci il nome del pacchetto da cercare: "
    read package
    echo "Eseguendo 'apt search $package'..."
    apt search $package
    ;;
```

remove Rimuove un pacchetto installato dal sistema.

```
"remove")
    echo -n "Inserisci il nome del pacchetto da rimuovere: "
    read package
    echo "Eseguendo 'sudo apt remove $package'..."
    sudo apt remove -y $package
    ;;
```

2.2.3 Sistema

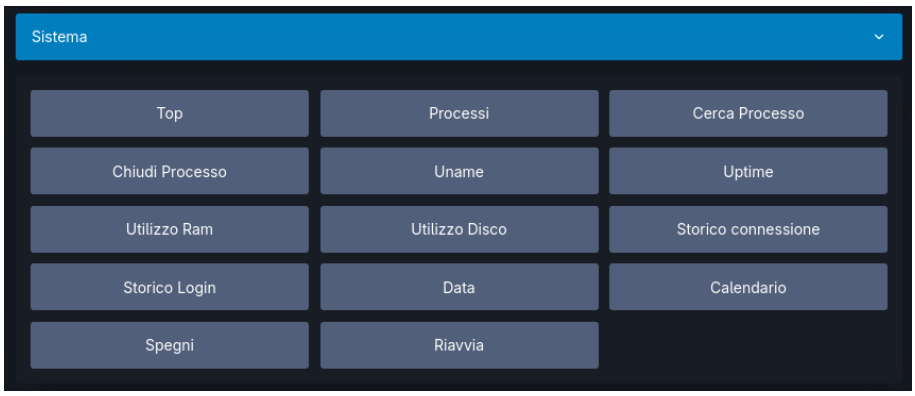

Figure 4: Interfaccia dei Comandi di Sistema

top Avvia il tool interattivo `top`, che fornisce una vista in tempo reale dei processi in esecuzione e del consumo di risorse.

```
"top")
    echo "Eseguendo 'top': Mostra i processi in esecuzione..."
    top
    ;;
```

ps Mostra i processi attivi in una struttura ad albero (`-forest`) per visualizzare le relazioni padre-figlio.

```
"ps")
    echo "Eseguendo 'ps aux --forest': Mostra i processi..."
    ps aux --forest
    ;;
```

searchps Cerca un processo specifico tramite `grep`. Richiede interattivamente il nome all'utente.

```
"searchps")
    echo "Eseguendo 'ps aux | grep \$name': Mostra i processi..."
    read -p "Nome del processo da cercare: " psname
    ps aux | grep $psname
    ;;
```

kill Termina un processo. Richiede interattivamente il Process ID (PID) da terminare.

```
"kill")
    echo -n "Inserisci l'ID del processo da terminare: "
    read pid
    echo "Eseguendo 'kill $pid': Termina il processo..."
    kill $pid
    ;;
```

uname Fornisce informazioni dettagliate sul sistema, inclusa la versione del kernel, il nome host e l'architettura.

```
"uname")
    echo "Eseguendo 'uname -a': Mostra le informazioni di sistema."
    uname -a
    ;;
```

uptime Mostra da quanto tempo il sistema e in esecuzione in un formato leggibile e amichevole.

```
"uptime")
    echo "Eseguendo 'uptime -p': Mostra da quanto tempo il sistema e attivo."
    uptime -p
    ;;
```

free Mostra l'utilizzo della memoria RAM e dell'area di swap in un formato leggibile (human-readable).

```
"free")
    echo "Eseguendo 'free -h': Mostra l'utilizzo della memoria..."
    free -h
    ;;
```

du Mostra l'utilizzo dello spazio su disco per la directory corrente in formato leggibile.

```
"du")
    echo "Eseguendo 'du -h': Mostra l'utilizzo dello spazio su disco..."
    du -h
    ;;
```

w Mostra chi e attualmente loggato nel sistema e cosa sta facendo.

```
"w")
    echo "Eseguendo 'w': Mostra chi e loggato."
    w
    ;;
```

last Mostra un elenco degli ultimi accessi (login) al sistema.

```
"last")
    echo "Eseguendo 'last -n 10': Mostra gli ultimi 10 login."
    last -n 10
    ;;
```

date Mostra la data e l'ora correnti del sistema.

```
"date")
    echo "Eseguendo 'date': Mostra la data e l'ora correnti."
    date
    ;;
```

cal Mostra un semplice calendario del mese corrente.

```
"cal")
    echo "Eseguendo 'cal': Mostra il calendario."
    cal
    ;;
```

shutdown Arresta il sistema. Tipicamente richiede privilegi di superutente.

```
"shutdown")
    echo "Eseguendo 'shutdown': Arresta il sistema."
    shutdown
    ;;
```

reboot Riavvia il sistema. Tipicamente richiede privilegi di superutente.

```
"reboot")
    echo "Eseguendo 'reboot': Riavvia il sistema."
    reboot
    ;;
```

2.2.4 File System

Figure 5: Interfaccia dei Comandi per il File System

inode Elenca i file e le directory nella posizione corrente, mostrando il loro numero di inode, utile per diagnosi a basso livello del file system.

```
"inode")
    echo "Eseguendo 'ls -li': Elenca i file... con gli inode."
    ls -li
    ;;
```

mkdir Crea una nuova directory. Il nome viene richiesto interattivamente.

```
"mkdir")
    echo -n "Inserisci il nome della cartella da creare: "
    read folder_name
    echo "Eseguendo 'mkdir $folder_name'..."
    mkdir $folder_name
    ;;
```

touch Crea un file vuoto, utile per testare permessi o per script che necessitano della preesistenza di un file.

```
"touch")
    echo -n "Inserisci il nome del file da creare: "
    read filename
    echo "Eseguendo 'touch $filename'..."
    touch $filename
    ;;
```

tree Mostra la struttura delle directory in formato ad albero, con una profondità specificata dall'utente.

```
"tree")
    read -p "Fino a che livello vuoi scendere? " n
    if [[ -z "$n" ]]; then
      tree -L 1
    fi
    tree -L $n
    ;;
```

ln Crea un link simbolico (symlink) da un file sorgente a una destinazione.

```
"ln")
    echo -n "Inserisci il file da linkare simbolicamente: "
    read source
    echo -n "Inserisci il percorso di destinazione: "
    read dest
    echo "Eseguendo 'ln -s $source $dest'..."
    ln -s $source $dest
    ;;
```

umask Comando educativo che spiega il funzionamento della umask e permette di impostarne un nuovo valore per la sessione corrente.

```
"umask")
    # ... (codice che mostra guida e chiede input)
    umask "$mask"
    ;;
```

chmod Guida interattiva per la modifica dei permessi di un file, spiegando il formato ottale prima di eseguire il comando.

```
"chmod")
    # ... (codice che mostra guida e chiede input)
    chmod $permissions "$filename"
    ;;
```

chown Cambia il proprietario e/o il gruppo di un file o di una directory. Richiede privilegi di superutente.

```
"chown")
    # ... (codice che chiede input)
    sudo chown -R $owner:$group $filename
    ;;
```

find Cerca file all'interno di un dato percorso, basandosi sul nome.

```
"find")
    echo -n "Inserisci il percorso in cui cercare: "
    read path
    echo -n "Inserisci il nome del file da cercare: "
    read filename
    echo "Eseguendo 'find $path -name $filename'..."
    find $path -name $filename
    ;;
```

file Determina il tipo di un file (es. testo ASCII, eseguibile, immagine).

```
"file")
    echo -n "Inserisci il nome del file: "
    read filename
    echo "Eseguendo 'file $filename'..."
    file $filename
    ;;
```

2.2.5 Gestione Utenti

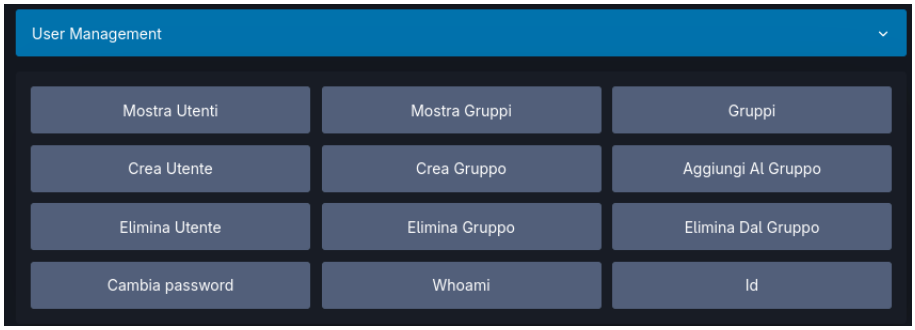

Figure 6: Interfaccia di Gestione Utenti

showusers Mostra tutti gli utenti non di sistema, filtrando per UID >= 1000 dal file `/etc/passwd`.

```
"showusers")
    echo "Utenti del sistema con UID >= 1000..."
    awk -F: '($3 >= 1000) {print $1}' /etc/passwd
    ;;
```

showgroups Mostra tutti i gruppi non di sistema, filtrando per GID >= 1000 dal file `/etc/group`.

```
"showgroups")
    echo "Gruppi del sistema con UID >= 1000..."
    awk -F: '($3 >= 1000) {print $1}' /etc/group
    ;;
```

groups Mostra i gruppi a cui l'utente corrente appartiene.

```
"groups")
    echo -n "Gruppi a cui appartiene $(whoami): "
    groups
    ;;
```

useradd Crea un nuovo utente nel sistema, creando la sua home directory (`-m`) e impostando `/bin/bash` come shell di default (`-s`). Richiede poi di impostare una password.

```
"useradd")
    echo -n "Inserisci il nome utente da creare: "
    read username
    echo "Eseguendo 'sudo useradd -m -s /bin/bash $username'..."
    sudo useradd -m -s /bin/bash $username
    echo "Ora imposta la password per $username:"
    sudo passwd $username
    ;;
```

groupadd Crea un nuovo gruppo di utenti.

```
"groupadd")
    echo -n "Inserisci il nome del gruppo da creare: "
    read group_name
    echo "Eseguendo 'sudo groupadd $group_name'..."
    sudo groupadd $group_name
    ;;
```

user-add-to-group Aggiunge un utente esistente a un gruppo esistente.

```
"user-add-to-group")
    echo -n "Inserisci il nome dell'utente da aggiungere al gruppo: "
    read user
    echo -n "Inserisci il nome del gruppo: "
    read group
    echo "Eseguendo 'sudo usermod -aG $group $user'..."
    sudo usermod -aG $group $user
    ;;
```

userdel Elimina un utente dal sistema, forzando la rimozione (`-f`) e cancellando la sua home directory (`-r`).

```
"userdel")
    echo -n "Inserisci il nome utente da eliminare: "
    read username
    echo "Eseguendo 'sudo userdel -fr $username'..."
    sudo userdel -fr $username
    ;;
```

groupdel Elimina un gruppo di utenti.

```
"groupdel")
    echo -n "Inserisci il nome del gruppo da eliminare: "
    read group_name
    echo "Eseguendo 'sudo groupdel $group_name'..."
    sudo groupdel $group_name
    ;;
```

user-remove-from-group Rimuove un utente da un gruppo specifico.

```
"user-remove-from-group")
    echo -n "Inserisci il nome dell'utente da rimuovere: "
    read user
    echo -n "Inserisci il nome del gruppo: "
    read group
    echo "Eseguendo 'gpasswd -d $user $group'..."
    sudo gpasswd -d $user $group
    ;;
```

passwd Permette all'utente corrente di cambiare la propria password.

```
"passwd")
    echo "Passwd permette di cambiare la password dell'utente corrente."
    passwd
    ;;
```

whoami Mostra il nome dell'utente attualmente loggato.

```
"whoami")
    echo "Eseguendo 'whoami': Mostra l'utente corrente."
    whoami
    ;;
```

id Mostra l'identità dell'utente corrente, inclusi UID, GID e i gruppi di appartenenza.

```
"id")
    echo "Eseguendo 'id': Mostra l'ID dell'utente e del gruppo."
    id
    ;;
```

2.2.6 Network

ping Invia pacchetti ICMP a un host per verificarne la raggiungibilità e misurare la latenza.

```
"ping")
    echo -n "Inserisci l'host da pingare: "
    read host
    echo "Eseguendo 'ping $host'..."
    ping $host
    ;;
```

wget Scarica file da un URL specificato.

```
"wget")
    echo -n "Inserisci l'URL da scaricare: "
    read url
    echo "Eseguendo 'wget $url'..."
    wget $url
    ;;
```

curl Trasferisce dati da o verso un server, usando vari protocolli. Utile per testare API.

```
"curl")
    echo -n "Inserisci l'URL da scaricare: "
    read url
    echo "Eseguendo 'curl $url'..."
    curl $url
    ;;
```

ip Comando moderno per visualizzare e manipolare routing, dispositivi, policy e tunnel.

```
"ip")
    echo "Eseguendo 'ip a': Mostra le informazioni sull'interfaccia di rete."
    ip a
    ;;
```

ifconfig Comando legacy per la configurazione delle interfacce di rete.

```
"ifconfig")
    echo "Eseguendo 'ifconfig': Mostra le informazioni sull'interfaccia di rete."
    ifconfig
    ;;
```

dig (Domain Information Groper) Strumento per interrogare i server DNS.

```
"dig")
    echo -n "Inserisci il dominio da interrogare: "
    read domain
    echo "Eseguendo 'dig $domain'..."
    dig $domain
    ;;
```

host Utility semplice per effettuare ricerche DNS.

```
"host")
    echo -n "Inserisci il dominio o l'indirizzo IP da interrogare: "
    read host
    echo "Eseguendo 'host $host'..."
    host $host
    ;;
```

hosts-edit Apre il file `/etc/hosts` con `vi` per modificare la risoluzione statica dei nomi host.

```
"hosts-edit")
    sudo vi /etc/hosts
    ;;
```

netstat Mostra le connessioni di rete, le tabelle di routing e altre statistiche. In questo caso, filtra per porta.

```
"netstat")
    echo "Inserisci una porta: "
    read port
    sudo netstat -anp | grep "$port"
    ;;
```

route Mostra/manipola la tabella di routing IP.

```
"route")
    echo "Eseguendo 'netstat -r': Mostra la tabella di routing IP."
    netstat -r
    ;;
```

arp Mostra/manipola la cache ARP del sistema.

```
"arp")
    echo "Eseguendo 'arp -a': Mostra la tabella ARP."
    arp -a
    ;;
```

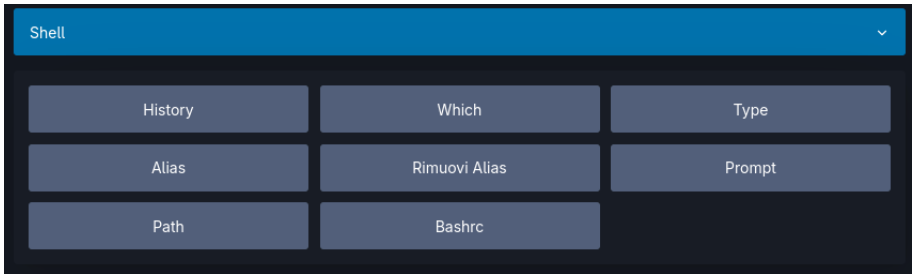

Figure 7: Interfaccia dei Comandi Shell

2.2.7 Shell

cd Cambia la directory corrente. Questo comando è fondamentale per la navigazione nel file system.

```
"cd")
    echo -n "Inserisci il percorso: "
    read path
    cd $path
    ;;
```

ls Elenca i contenuti della directory corrente, mostrando file e altre directory.

```
"ls")
    ls -la
    ;;
```

pwd (Print Working Directory) Mostra il percorso completo della directory in cui ci si trova attualmente.

```
"pwd")
    pwd
    ;;
```

history Mostra un elenco degli ultimi comandi eseguiti nella sessione corrente.

```
"history")
    history
    ;;
```

exit Termina la sessione corrente della shell.

```
"exit")
    exit
    ;;
```

2.2.8 Hardware

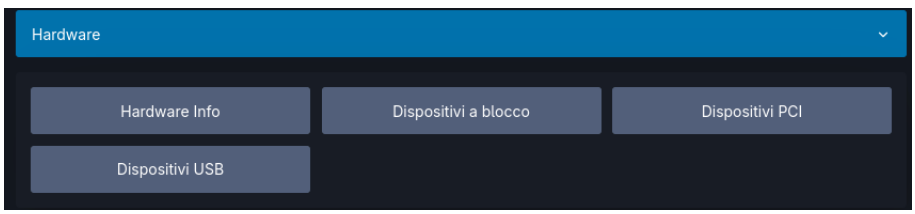

Figure 8: Interfaccia dei Comandi Hardware

lscpu Mostra informazioni dettagliate sull'architettura della CPU.

```
"lscpu")
    lscpu
    ;;
```

lsusb Elenca i dispositivi USB collegati al sistema.

```
"lsusb")
    lsusb
    ;;
```

lspci Elenca tutti i dispositivi PCI.

```
"lspci")
    lspci
    ;;
```

lshw Elenca informazioni dettagliate sull'hardware del sistema.

```
"lshw")
    sudo lshw
    ;;
```

lsblk Elenca i dispositivi a blocchi, come dischi e partizioni.

```
"lsblk")
    lsblk
    ;;
```

2.2.9 Kernel

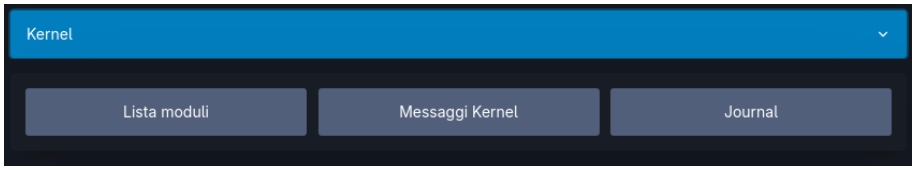

Figure 9: Interfaccia dei Comandi Kernel

dmesg Stampa o controlla il ring buffer del kernel.

```
"dmesg")
    dmesg
    ;;
```

lsmod Mostra lo stato dei moduli nel Kernel Linux.

```
"lsmod")
    lsmod
    ;;
```

modinfo Mostra informazioni su un modulo del Kernel Linux.

```
"modinfo")
    echo -n "Inserisci il nome del modulo: "
    read module
    modinfo $module
    ;;
```

3 Conclusioni

MeShell è un eccellente esempio di come le tecnologie web moderne possano essere sfruttate per creare potenti strumenti di amministrazione di sistema. L'analisi del codice rivela un'architettura ben pensata: il backend Go gestisce la complessità del protocollo SSH e del PTY; il frontend JavaScript con Xterm.js offre un'esperienza utente fluida e nativa; lo script Bash astrae i comandi, rendendoli accessibili e riducendo la possibilità di errori. Questa sinergia tra componenti eterogenei crea uno strumento flessibile e potente, ideale per contesti didattici e per l'amministrazione rapida di sistemi.