



UNIVERSITATEA DIN CRAIOVA  
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI  
ELECTRONICĂ  
DEPARTAMENTUL DE AUTOMATICĂ ȘI ELECTRONICĂ



# **MANAGEMENTUL PROIECTELOR SOFTWARE**

## **Control aplicație desktop via WebSockets**

### **MASTERANZI:**

IVAN MIHAELA DIANA  
ILINA TEDDY RĂZVAN IONUȚ

### **PROFESOR COORDONATOR:**

BOGDAN HUREZEANU

**GRUPA:** SAI II B

CRAIOVA,  
2022

# Cuprins

1.	VERSIUNILE DOCUMENTULUI .....	3
2.	SCOPUL DOCUMENTULUI .....	3
3.	DESCRIERE PROIECT .....	3
4.	SPECIFICAȚII .....	5
5.	IMPLEMENTARE .....	5

## 1. VERSIUNILE DOCUMENTULUI

### Revizii

Versiune	Autor	Descrierea versiunii
1.1	Ivan Mihaela	Crearea versiunii initiale a documentului
1.2	Ivan Mihaela	Adaugarea specificațiilor de proiect
1.3	Ilina Teddy	Completarea campului „Descriere proiect”
1.4	Ivan Mihaela	Adaugarea modului de implementare
1.5	Ilina Teddy	Adaugarea diagramei UML

## 2. SCOPUL DOCUMENTULUI

A fost realizat de echipa formată din Ivan Mihaela-Diana și Ilina Teddy-Răzvan-Ionuț în cadrul proiectului la disciplina *Managementul proiectelor software*. Scopul acestui document este de a prezenta specificațiile și modul de realizare al proiectului *Automatizare aplicație desktop cu ajutorul AHK și control via WebSockets*.

## 3. DESCRIERE PROIECT

Majoritatea serviciilor pe care le folosim în prezent sunt bazate pe web. Folosirea acestora este ușoară deoarece utilizatorii nu au nimic de instalat și pot începe să folosească produsul cu doar câteva click-uri. De asemenea, dezvoltarea, mentenanța și actualizarea acestora fiind ușoară pentru dezvoltatori. Datorită evoluției tehnologiilor web din ultimii ani, putem avea într-adevăr interfețe interactive. Cu toate acestea, în unele cazuri de utilizare, este încă necesară o aplicație desktop, de exemplu, pentru serviciile care trebuie să acceseze fișierele locale pentru a le sincroniza sau o aplicație care trebuie să acceseze dispozitivele locale, cum ar fi GPS-ul, pentru a actualiza harta, date despre dispozitiv cum ar fi memoria RAM utilizată, cât la sută este utilizat procesorul.

Pentru ca acest lucru să funcționeze, este necesară dezvoltarea unei aplicații cu o interfață desktop, diferită și separată de cea web, dar care va fi greu de întreținut. Soluția ideală ar fi realizarea unui proces daemon (proces de fundal care răspunde solicitărilor de servicii) care oferă

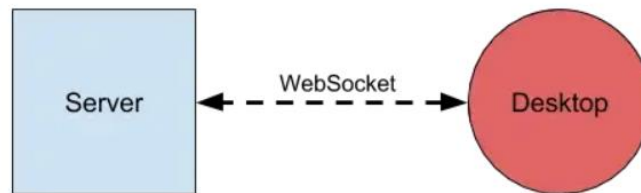
acces la computerul local, pe care l-am putea controla din serviciul web. Problema aici este de a face procesul desktop capabil să poată trimite și primi date către și de la serviciul nostru web.

În mod tradițional, un client face cereri către un server în mod sincron: acesta completează datele într-un formular și primește un răspuns în schimb.



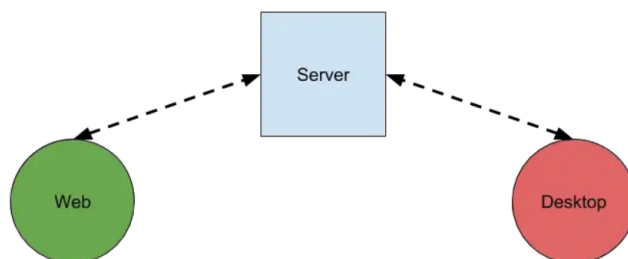
Desktop making HTTP requests to server

Mai departe, problema constă în a face serverul să trimită cereri către desktop. Pentru asta, vom folosi WebSockets. WebSockets este un protocol standard web care urmărește deschiderea unui canal de comunicație full-duplex între un client și un server. Cu alte cuvinte, deschide un tunel în care corespondenții pot trimite și primi mesaje.



WebSocket between server and desktop

Pentru interfața web, putem deschide doar un WebSocket între browser și server; apoi, serverul va redirecționa mesajele din browser către aplicația desktop și invers.



Server making the bridge between the WebSockets of web and desktop

## 4. SPECIFICAȚII

**S100:** Serverul acceptă doar conexiuni WebSockets și va interconecta un client web și un client desktop printr-un client\_id și va transmite mesajele între ei (canal de comunicație full-duplex).

**S110:** Dacă interconectarea dintre clientul web și clientul desktop printr-un client\_id se va efectua cu succes, pe interfața web se va afișa statusul *CONECTAT*. Altfel, statusul afișat va fi *DECONNECTAT*.

**S120:** Desktop-ul va raporta utilizarea procesorului computerului (computer CPU usage) și procentul de utilizarea al memoriei RAM (computer RAM memory usage) în fiecare secundă către server.

**S130:** Serverul va redirecționa mesajele primite de la aplicația desktop către interfața web și va afișa datele primite .

**S140:** Web-ul va afișa procentele de utilizare ale CPU-ului și a memoriei RAM pe care le primește de la desktop și va conține un buton care odată apăsă va emite un mesaj sonor către desktop (o informare că mesajele au fost recepționate cu succes).

## 5. IMPLEMENTARE

### SERVER:

În crearea unui WebSocket s-a folosit setul de instrumente Starlette care este ideal pentru construirea de servicii web asincrone în Python. După ce am stabilit conexiunea, ne așteptăm ca clientul să ne trimită un mesaj de „autentificare”, astfel încât să îl putem potrivi cu celălalt client.

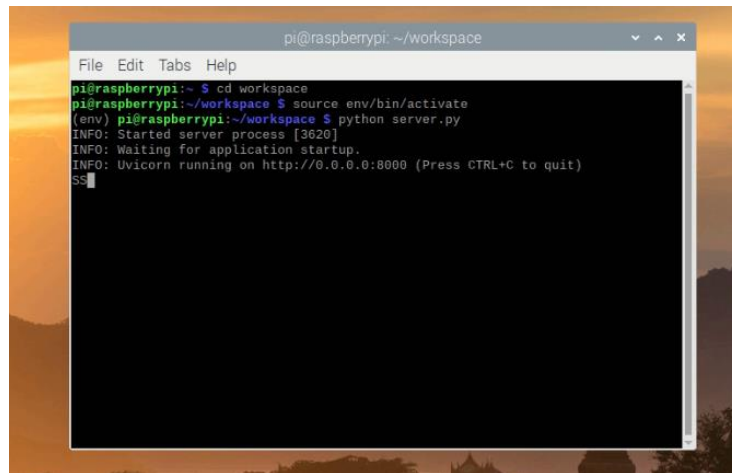
```
@app.websocket_route('/ws')
async def websocket_endpoint(websocket):
    await websocket.accept()

    # mesaj de autentificare print-un client id
    message = await receive_json(websocket)
    client_mode = message['client_mode']
    client_id = message['client_id']
    websockets[client_mode][client_id] = websocket
```

Păstrăm websocket-ul în memorie, astfel încât să putem difuza mesaje în el:

```
try:
    # transmite catre mirror client
    await websockets[mirror_mode][client_id].send_text(
        json.dumps(message)
    )
except KeyError:
    logger.debug(
        f'Client {client_id}[{mirror_mode}] neconectat'
```

Pornim serverul si asteptam conectarea unui client:



## DESKTOP:

Desktopul folosește API-ul *asyncio* pentru a raporta utilizarea procesorului si a memoriei în fiecare secundă și reacționează la mesajele beep transmise din interfața web prin intermediul serverului.

```
async def handler(uri, client_id):
    async with websockets.connect(uri) as websocket:
        message = {
            'event': 'authentication',
            'client_id': client_id,
            'client_mode': 'desktop'
        }
        await websocket.send(json_to_payload(message))

        consumer_task = asyncio.ensure_future(
            consumer_handler(websocket))
        producer_task = asyncio.ensure_future(
            cpu_usage_reporter(websocket))
        producer_task2 = asyncio.ensure_future(
            ram_usage_reporter(websocket))
        done, pending = await asyncio.wait(
            [consumer_task, producer_task, producer_task2],
            return_when=asyncio.FIRST_COMPLETED,
        )
        for task in pending:
            task.cancel()
```

Pentru a transmite informatii pentru a fi afisate de catre clientul web, se vor folosi functii continute de biblioteca psutil: *psutil.cpu\_percent()* pentru CPU usage si *psutil.virtual\_memory()* pentru RAM memory usage. Datele sunt transmise prin intermediul metodei *websocket.send()*. Pornirea aplicatiei desktop se face prin intermediul comenzii *python desktop.py client*, unde client este *client\_id*.

```

# raporteaza catre web procentul de utilizare CPU, este folosita functia cpu_usage_reporter() continuta de biblioteca psutil
async def cpu_usage_reporter(websocket):
    psutil.cpu_percent()
    while (True):
        await asyncio.sleep(1)
        message = {
            'event': 'cpu',
            'value': psutil.cpu_percent(),
        }
        await websocket.send(json_to_payload(message))
        logger.debug(f'Transmite CPU usage catre server: {message}')

# raporteaza catre web procentul de utilizare RAM, este folosita functia virtual_memory() continuta de biblioteca psutil
async def ram_usage_reporter(websocket):
    psutil.virtual_memory()
    while (True):
        await asyncio.sleep(1)
        message = {
            'event': 'ram',
            'value': psutil.virtual_memory().percent,
        }
        await websocket.send(json_to_payload(message))
        logger.debug(f'Transmite RAM usage catre server: {message}')

```

Datele sunt transmise cu succes de la desktop catre server si sunt gata pentru a fi transmise prin intermediul websocket catre interfața web de îndată ce clientul web se va conecta.

```

pi@raspberrypi: ~/workspace
File Edit Tabs Help
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.4}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'cpu', 'value': 6.3}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.3}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'cpu', 'value': 8.2}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.3}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'cpu', 'value': 5.7}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.3}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'cpu', 'value': 5.8}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.3}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'cpu', 'value': 5.1}
DEBUG: Client client[web] deconectat
DEBUG: Mesaj primit de la client[desktop]: {'event': 'ram', 'value': 33.3}
DEBUG: Client client[web] deconectat

pi@raspberrypi: ~/workspace
File Edit Tabs Help
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.5}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 14.9}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.5}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 7.9}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 15.1}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.5}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 11.6}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.5}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 8.1}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.5}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 6.0}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.4}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 6.3}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}
C i: _r n: e CPU usage {'event': 'cpu', 'value': 8.2}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 5.7}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 5.8}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}
DEBUG: __main__:Transmite CPU usage catre server: {'event': 'cpu', 'value': 5.1}
DEBUG: __main__:Transmite RAM usage catre server: {'event': 'ram', 'value': 33.3}

```

## WEB:

Conectarea clientului web la server se realizeaza prin deschiderea unui canal de comunicatie websocket. Datele transmise de aplicatia desktop sunt receptionate prin intermediul metodei `getElementById()` si afisate pe interfata web.

```
const createSocket = (clientId) => {  
  const socket = new WebSocket("ws://localhost:8000/ws");  
  
  socket.onopen = (event) => {  
    socket.send(JSON.stringify({  
      event: 'authentication',  
      client_id: clientId,  
      client mode: 'web',  
    }));  
  }  
}
```

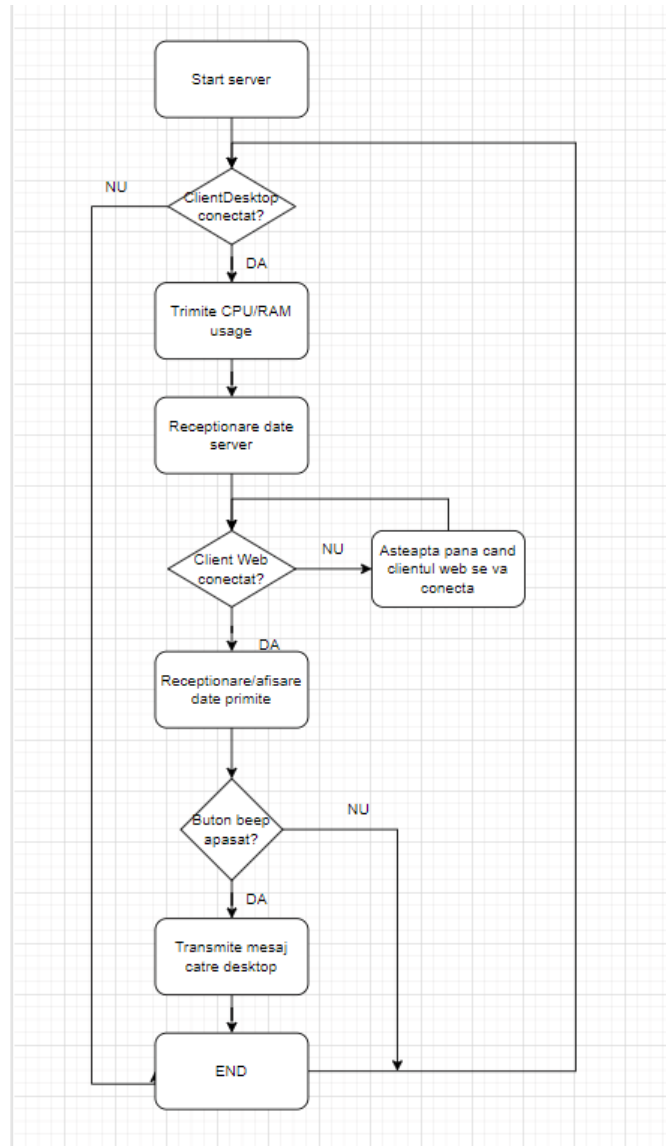
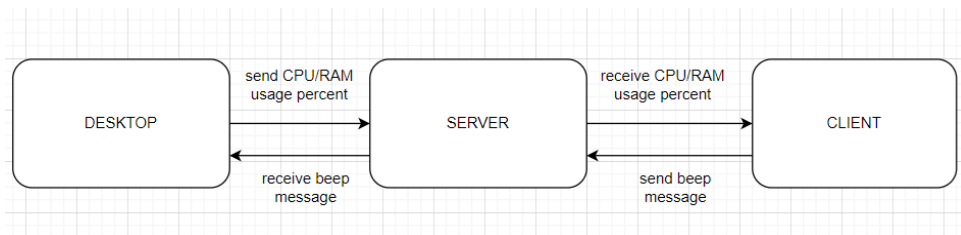
Interfata web contine si un buton care transmite un mesaj sonor *beep* catre aplicatia desktop. Comunicarea prin intermediul WebSockets s-a realizat cu succes, datele fiind transmise cu succes intre cei doi clienti(client desktop si client web).

The screenshot displays a web browser window titled "web.html - Chromium" showing a web interface for "Control aplicație desktop via WebSockets". The interface indicates a successful connection with the status "Status: CONECTAT" and "Client ID: client". It also shows system metrics: "CPU usage: 9.3% RAM usage: 33%". A "Beep" button is visible, and a "client" input field is present. Below the input field, there are two buttons: "CONECTARE" (green) and "DECONECTARE" (red). The interface also features a diagram illustrating the connection between a desktop client, a server, and a web client.

The terminal window on the right shows the output of the application, displaying a series of debug messages. The messages indicate the successful establishment of a WebSocket connection and the transmission of data between the desktop client and the web client. The messages are as follows:

```
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 19.8 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.3 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 14.4 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.2 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 9.7 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.2 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 9.5 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.1 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 14.9 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.6 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 23.4 }  
DEBUG: Mesaj primit de la client[web]: { 'event': 'beep' }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 20.1 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.3 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 23.2 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.5 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 14.7 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.2 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 8.2 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.1 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'cpu', 'value': 9.3 }  
DEBUG: Mesaj primit de la client[desktop]: { 'event': 'ram', 'value': 33.6 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.3 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 14.4 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.2 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 9.7 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.2 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 9.5 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.1 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 14.9 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.6 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 23.4 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.2 }  
DEBUG: _main_Mesaj server primit: { 'event': 'beep' }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 20.1 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.3 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 23.2 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.5 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 14.7 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.2 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 8.2 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.1 }  
DEBUG: _main_Transmite CPU usage catre server: { 'event': 'cpu', 'value': 9.3 }  
DEBUG: _main_Transmite RAM usage catre server: { 'event': 'ram', 'value': 33.6 }
```

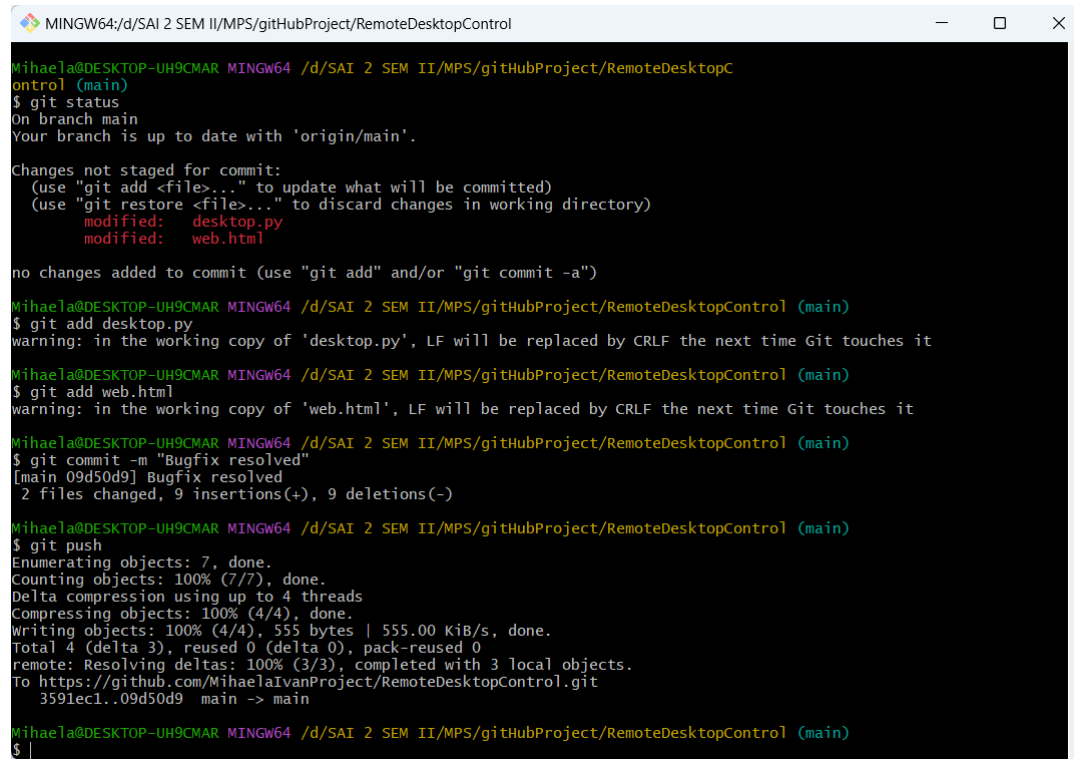




Proiectul se gaseste pe gitHub la adresa web:

<https://github.com/MihaelaIvanProject/RemoteDesktopControl>

Adaugarea/stergera sau actualizarea fisierelor s-a realizat prin intermediul interfetei desktop git prin comenzi. Un exemplu se poate vedea in imaginea de mai jos.



```
MINGW64:/d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl
Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   desktop.py
        modified:   web.html

no changes added to commit (use "git add" and/or "git commit -a")

Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$ git add desktop.py
warning: in the working copy of 'desktop.py', LF will be replaced by CRLF the next time Git touches it

Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$ git add web.html
warning: in the working copy of 'web.html', LF will be replaced by CRLF the next time Git touches it

Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$ git commit -m "Bugfix resolved"
[main 09d50d9] Bugfix resolved
 2 files changed, 9 insertions(+), 9 deletions(-)

Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 555 bytes | 555.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/MihaelaIvanProject/RemoteDesktopControl.git
   3591ec1..09d50d9  main -> main

Mihaela@DESKTOP-UH9CMAR MINGW64 /d/SAI 2 SEM II/MPS/gitHubProject/RemoteDesktopControl (main)
$
```