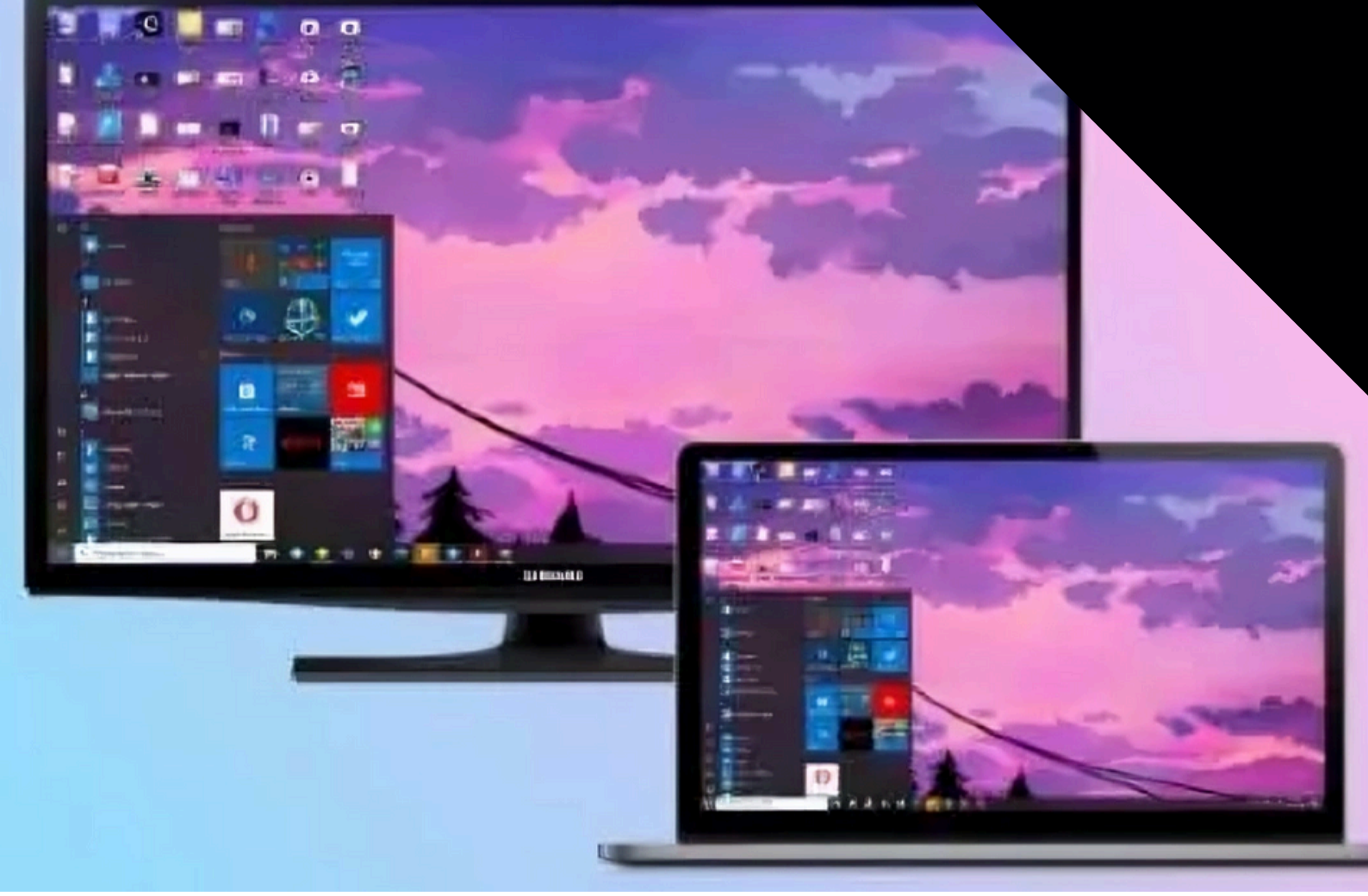


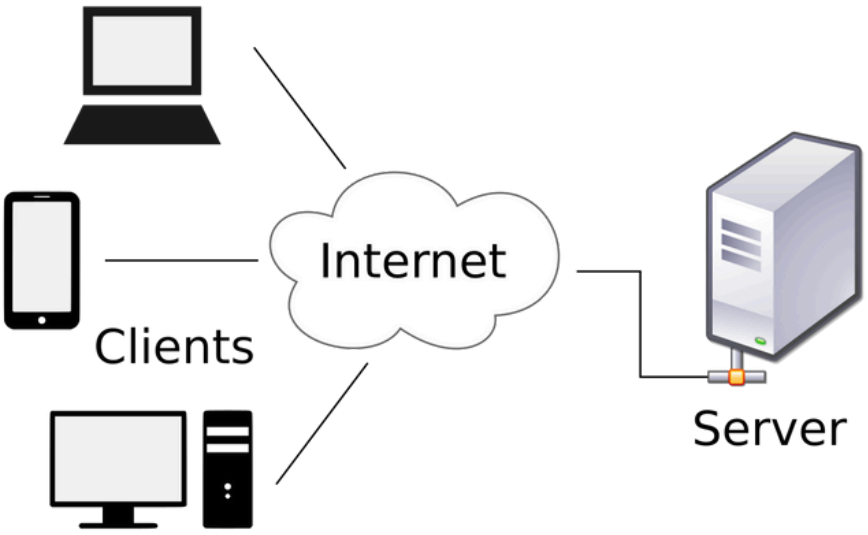
PCS-2 Project Report

SpaceLink

Prepared by

Aditya Rathor(B22AI044)
Vishesh Sachdeva(B22AI050)

ABOUT THE PROJECT

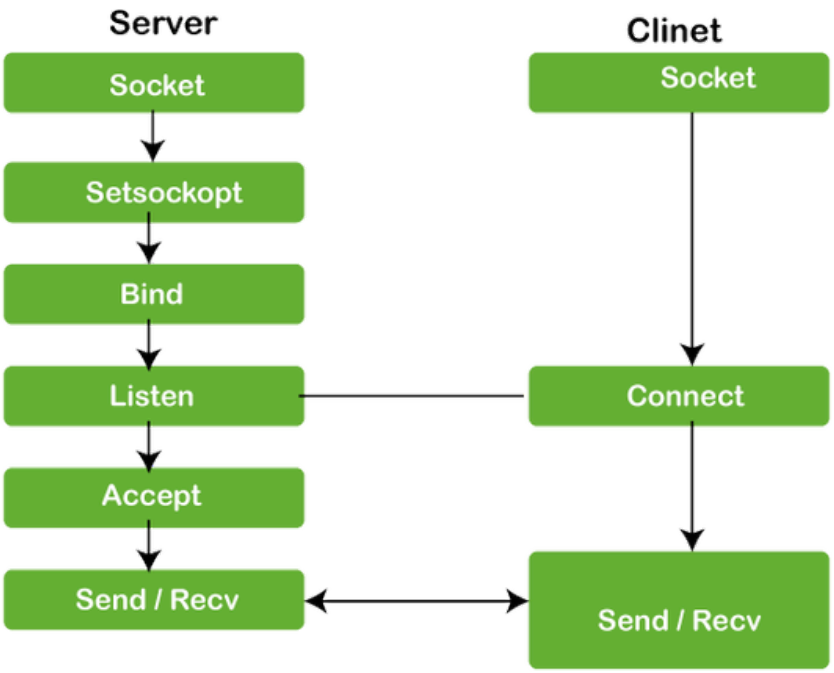


We implemented a server-client architecture for remote mouse control and screen sharing. The server runs on a host machine while the client connects to the server.
The client can:

- send commands to control the mouse remotely
- receive screenshots of the server's screen.
- And can also control mouse of the server screen using his mouse based on synchronous transfer of position of the mouse from client to server

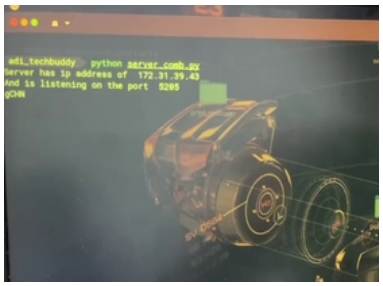
COMPONENTS USED:

- **Socket Programming:** Both server and client communicate over sockets, allowing bidirectional data transfer.
- **Threading**: Utilizes threads to handle simultaneous tasks such as mouse control, screen pixels transfer(60fps) and user input commands.
- **PyAutoGUI**: Library used for controlling the mouse cursor and performing actions like moving and clicking programmatically.
- **Pygame**: Employed for displaying screenshots on the client side and capturing mouse events for remote clicking.
- **mss** (Multi-Screen Shot): Used for capturing screenshots efficiently on the server side.

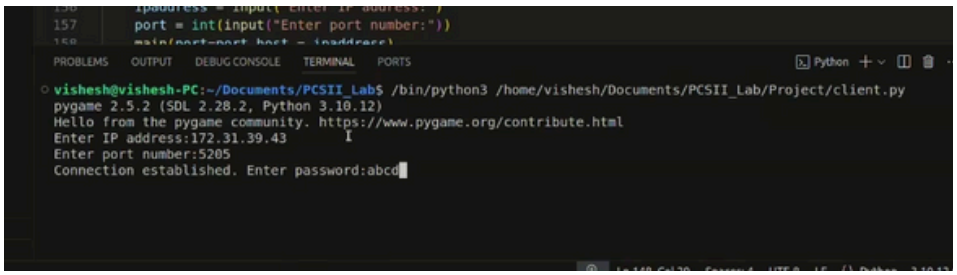


HOW IT WORKS?

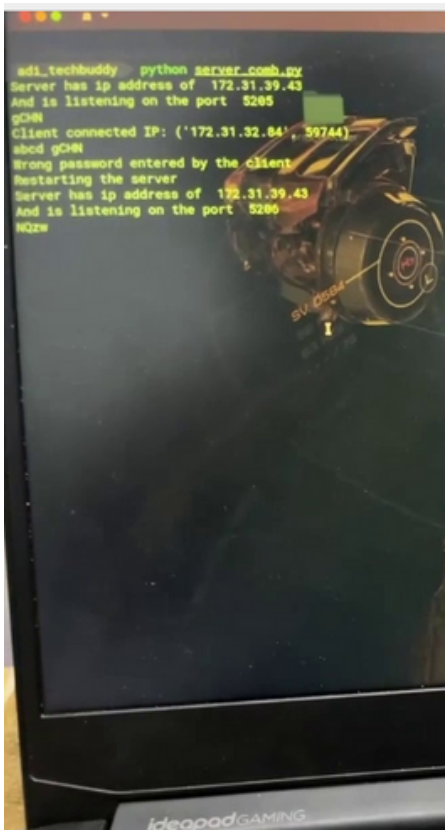
- **Authentication**:
  - Ensures secure communication by requiring clients to enter a randomly
  - generated password of 4 digit.
  - Is a user enters a wrong password then connection will be automatically ended
  - from the server side and the server program will rerun with different port
  - number and a newly randomly generated password.



Started server with the following ip and port



Invalid password entered by the client



Server restarted with new port and password

HOW IT WORKS?

- **Threading**: Since this project required handling multiple task concurrently We used the concept of threads.
  - Server Threads:
    - **THREAD FOR SCREEN SHARING (TAKE_SCREENSHOTS):**
      - This function is designed to run in a separate thread for each client that connects to the server.
      - Utilizes the **mss library** to capture the screen.
      - Defines a region (rect) to capture, covering the entire screen based on the specified width and height which is kept as **1920*1080** for best quality
      - Captures the screen continuously in a loop.
      - Compresses the captured screen image using zlib compression with a specified compression level set to 6.
      - Sends:
        - size of the compressed pixels length,
        - followed by the actual pixels length,
        - finally sends the compressed pixels data over the network to the client.
      -
    - **THREAD FOR MOUSE CONTROL (HANDLE_MOUSE_CONTROL):**
      - This function also similar to above runs in a separate thread for each client that connects to the server.
      - It continuously listens for messages from the client regarding mouse movements and clicks.
      - It interprets the received messages and performs corresponding actions using **pyautogui**.
      - The commands are as follows:
        - w : to move up
        - s : to move down
        - a: to move left
        - d: to move right
        - c: to click

        → By default moves by 10px but if value specified like : "w int": it will move by int pixels

      - The thread continues running until it receives a 'stop' message from the client, indicating that the client has disconnected.

HOW IT WORKS?

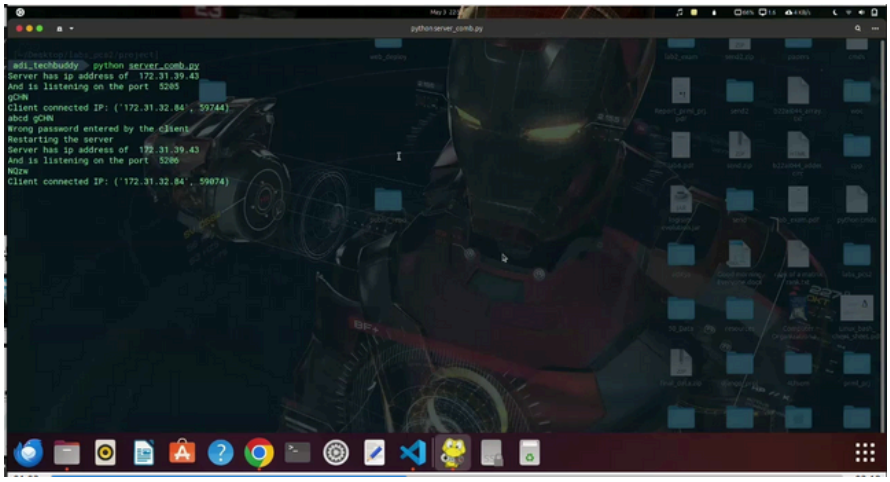
- **Client Threads:**
  - **THREAD FOR MOUSE COMMANDS (SEND_MOUSE_COMMANDS):**
    - This function runs in a separate thread.
    - It continuously waits for user input from the command line to control the mouse or to stop the program.
    - When the user enters '**ctrl**', it toggles the ctrl variable, indicating whether the client should acquire or release control over the server's mouse.
    - It sends the user's input commands to the server over the network.

  - **THREAD FOR RETRIEVING SCREENSHOTS (RETREIVE_SCREENSHOT):**
    - This function also runs in a separate thread and enters in a infinite loop for handling image information in continuous manner .
    - Retrieves the screen width (W) and height (H) using pygame.display.Info().
    - Sets up a pygame window with resizable dimensions matching the screen size so that that window becomes dynamic.
    - Receives the size of the compressed pixels length, data using the recvall function.
    - Decompresses the received compressed pixels data.
    - Creates a pygame surface from the raw pixel data.
    - Displays the screenshot on the pygame window, updating it at a rate of 60 frames per second

  - **THREAD FOR SENDING MOUSE POSITION (POSITION_FUN):**
    - Once ctrl is enabled for mouse control it continuously sends the client's mouse position to the server pyautogui.position to send the coordinates of the client mouse
    - It also sends a 'c' command if the client has clicked the mouse, indicating a mouse click event.
    - It runs in a loop until the user exits the program.

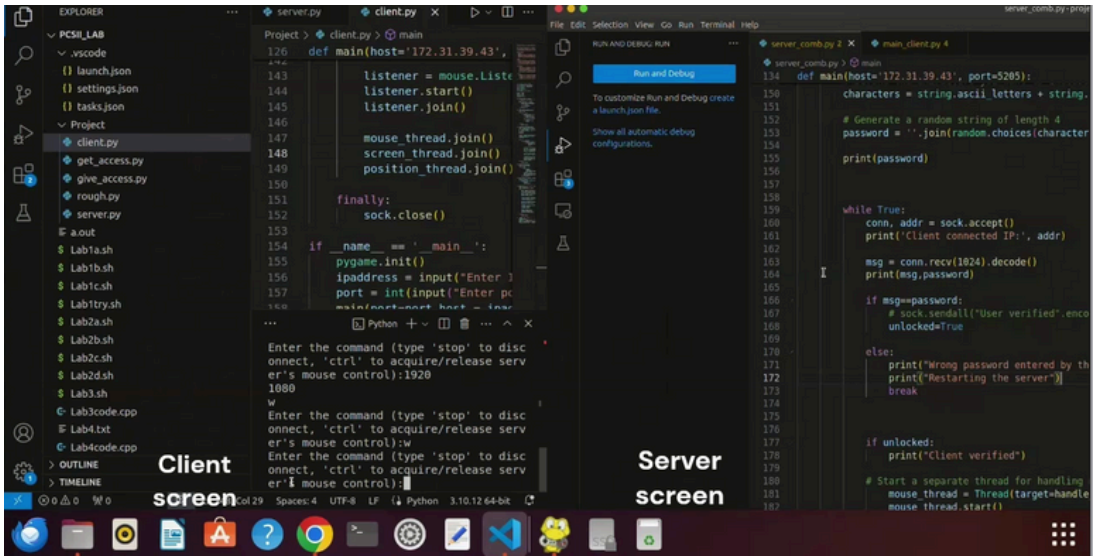
  - **LISTENER THREAD FOR MOUSE CLICKS (ON_CLICK AND LISTENER):**
    - This thread is started using the pynput.mouse.Listener class.
    - It listens for mouse click events on the client's machine.
    - When a left mouse click event is detected, it sets the clicked global variable to True(needed to inform the position_fun that it should start sending positions).

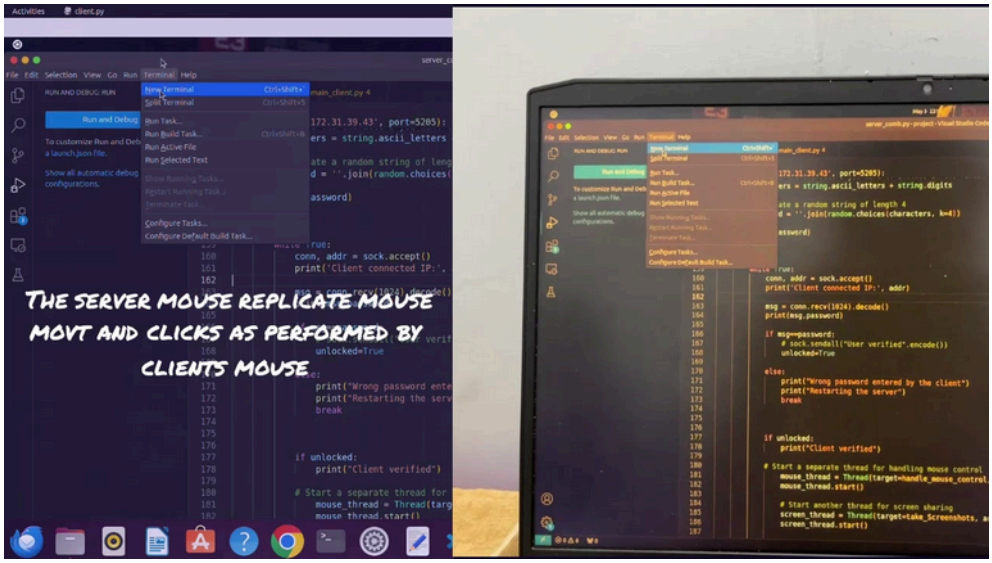
SCREENSHOTS



Started screensharing on client's screen

Commands entered on Terminal to control mouse

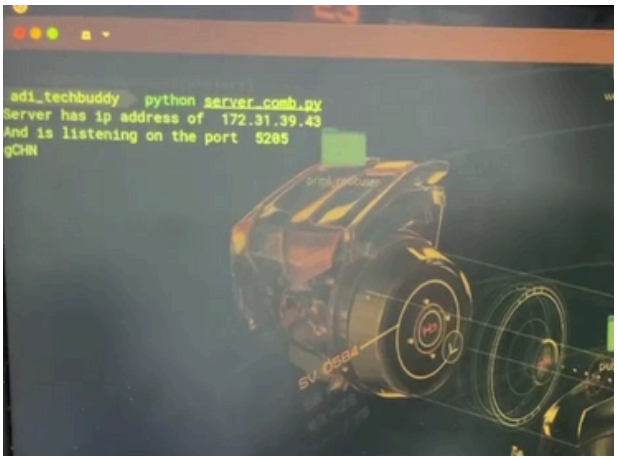


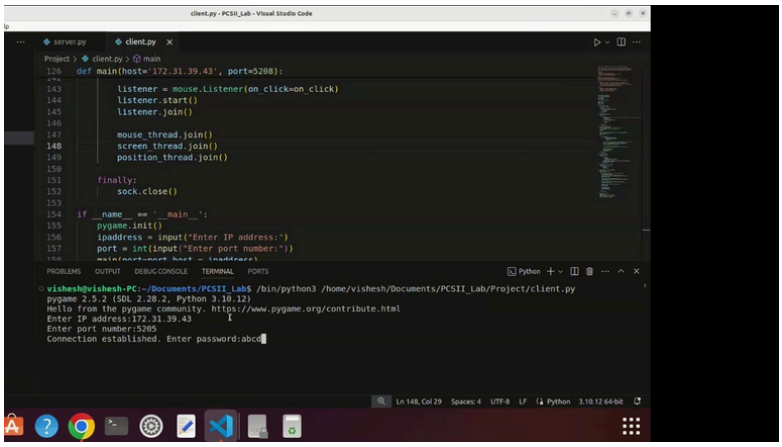


Getting Direct mouse control and controlling the server desktop using it

PIPELINE:

- **SERVER SIDE:**
  - Listens for incoming connections and authenticates clients using a randomly generated password.
  - Handles mouse control commands received from the client and performs corresponding actions on the server's mouse.
  - Captures screenshots of the server's screen and sends them to the client.





- **CLIENT SIDE:**
  - Connects to the server using the provided IP address and port number.
  - Sends authentication password to the server for verification.
  - Allows the user to input commands for mouse control and sends them to the server.
  - Receives and displays screenshots from the server.

CONCLUSION

The provided server-client mouse control and screen sharing application offer basic functionality for remote control and monitoring of a server's screen using socket programming.

Concept of Threading is used effectively to handle simultaneous tasks such as mouse control, screen capturing, and user input handling, ensuring smooth performance and responsiveness.

With further enhancements and refinements, it could serve as a valuable tool for remote administration and collaborative work environments.