DON BOSCO INSTITUTE OF TECHNOLOGY

Premier Automobiles Road, Kurla (W), Mumbai-70

Approved by AICTE, Govt. of Maharashtra

&

Affiliated to the University of Mumbai

S.E. MINI PROJECT REPORT
CSL405 -PYTHON PROGRAMMING

On

"Seamless File Sharing From PC to Mobile"

Department of Computer Engineering

University of Mumbai

April 2025

DON BOSCO INSTITUTE OF TECHNOLOGY

Premier Automobiles Road, Kurla (W), Mumbai-70

MINI PROJECT TITLE:- **SEAMLESS FILE SHARING FROM PC TO MOBILE**

INSTITUTE NAME:- DON BOSCO INSTITUTE OF TECHNOLOGY

INSTITUTE ADDRESS:- PREMIER AUTOMOBILES ROAD
KURLA (WEST) MUMBAI 400070

DEPARTMENT:- COMPUTER ENGINEERING

CLASS:- SE COMPS B

PROJECT GROUP MEMBERS:-

Name	Roll No
Ansari Mohammed Saif	01
Mohiuddin Merchant	16
Aaditi Bhatade	02
Mohd Hasshir	20

SIGNATURE OF INTERNAL GUIDE: [illegible]

INTERNAL GUIDE:- PROF.SHAINILA SHAIKH

INDEX

AIM OF THE PROJECT

The aim of this mini project is to develop a web-based file-sharing system that enables seamless and efficient file transfers between a PC and a mobile device using Python, Tkinter, and WebSockets. Traditional file-sharing methods, such as USB, Bluetooth, or cloud services, often suffer from speed limitations, external dependencies, and security risks. This project eliminates these challenges by establishing a real-time communication channel over a local Wi-Fi network, allowing users to access and download files effortlessly without requiring an active internet connection. The primary objectives of the project are as follows:

1. To design and implement a contactless, efficient alternative to traditional file-sharing methods.
2. To establish a real-time WebSocket-based communication channel between the server (PC) and client (mobile device).
3. To develop a user-friendly web interface for browsing and downloading available files.
4. To ensure secure and reliable file access while maintaining a smooth user experience.
5. To handle edge cases such as network interruptions and multiple concurrent client requests efficiently.

CODE:

SERVER-SIDE CODE:

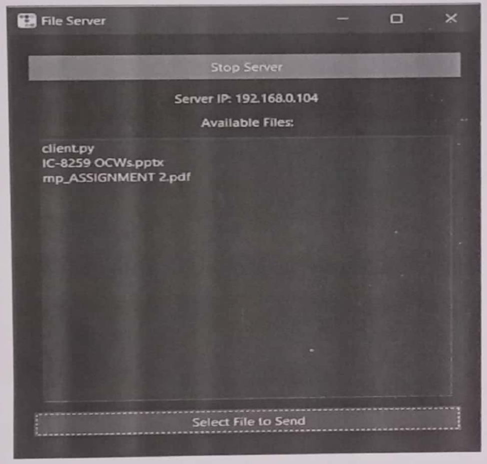
```
import tkinter as tk
from tkinter import ttk, filedialog
import threading
import asyncio
import websockets
import json
import os
import shutil
import socket
from base64 import b64encode
import ttkbootstrap as tb

class FileServer:
    def __init__(self):
        self.window = tb.Window(themename="darkly")
        self.window.title("File Server")
        self.window.geometry("500x600")

        # Get IP address
        self.server_ip = self.get_local_ip()
        self.server_port = 8765

        # Frame
        self.main_frame = ttk.Frame(self.window, padding=20)
        self.main_frame.pack(expand=True, fill=tk.BOTH)

        # Server Button
        self.server_btn = tb.Button(self.main_frame, text="Start Server", bootstyle="success",
command=self.toggle_server)
        self.server_btn.pack(pady=10, fill=tk.X)

        # Server Status Label
        self.status_label = ttk.Label(self.main_frame, text=f"Server IP: {self.server_ip}")
        self.status_label.pack(pady=5)

        # File List
        self.file_list_label = ttk.Label(self.main_frame, text="Available Files:")
        self.file_list_label.pack(pady=5)
        self.file_list = tk.Listbox(self.main_frame, width=50, height=10)
        self.file_list.pack(padx=10, pady=5, fill=tk.BOTH, expand=True)

        # Send File Button
        self.send_btn = tb.Button(self.main_frame, text="Select File to Send",
bootstyle="primary", command=self.select_file)
        self.send_btn.pack(pady=10, fill=tk.X)

        # Setup uploads directory
        self.upload_dir = "uploads"
        os.makedirs(self.upload_dir, exist_ok=True)
        self.update_file_list()

        self.server_thread = None
        self.server = None
        self.running = False

    def get_local_ip(self):
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            s.connect(("8.8.8.8", 80))
            ip = s.getsockname()[0]
            s.close()
            return ip
        except Exception as e:
            print("[ERROR] Unable to get local IP:", e)
            return "127.0.0.1"

    def update_file_list(self):
        self.file_list.delete(0, tk.END)
        for f in os.listdir(self.upload_dir):
            self.file_list.insert(tk.END, f)

    def select_file(self):
        file_path = filedialog.askopenfilename()
        if file_path:
            file_name = os.path.basename(file_path)
            destination = os.path.join(self.upload_dir, file_name)
            shutil.copy(file_path, destination)
            print(f"[INFO] File added: {file_name}")
            self.update_file_list()

    async def handle_client(self, websocket):
        print("[INFO] Client connected")
        files = os.listdir(self.upload_dir)
        await websocket.send(json.dumps({"type": "list", "files": files}))
        async for message in websocket:
            data = json.loads(message)
            if data["type"] == "download":
                filename = data["file"]
                filepath = os.path.join(self.upload_dir, filename)
                if os.path.exists(filepath):
                    with open(filepath, "rb") as f:
                        content = b64encode(f.read()).decode("utf-8")
                    await websocket.send(json.dumps({"type": "file", "name": filename, "content":
content}))
                    print(f"[INFO] Sent file: {filename}")

    async def run_server(self):
        print("[INFO] Server starting...")
        async with websockets.serve(self.handle_client, self.server_ip, self.server_port):
            await asyncio.Future()

    def start_server(self):
        self.running = True
        self.server_btn.config(text="Stop Server", bootstyle="danger")
        self.server_thread = threading.Thread(target=self.run_async_server, daemon=True)
        self.server_thread.start()
        print("[INFO] Server started.")

    def stop_server(self):
        self.running = False
        self.server_btn.config(text="Start Server", bootstyle="success")
        print("[INFO] Server stopped.")

        # Empty the uploads folder when the server stops
        self.clear_uploads_folder()
        self.update_file_list()

    def clear_uploads_folder(self):
        try:
            for file in os.listdir(self.upload_dir):
                file_path = os.path.join(self.upload_dir, file)
                if os.path.isfile(file_path):
                    os.remove(file_path)
                    print(f"[INFO] Deleted: {file}")
        except Exception as e:
            print("[ERROR] Failed to clear uploads folder:", e)

    def toggle_server(self):
        if self.running:
            self.stop_server()
        else:
            self.start_server()

    def run_async_server(self):
        asyncio.run(self.run_server())

    def run(self):
        self.window.mainloop()

if __name__ == "__main__":
    server = FileServer()
    server.run()
```

CLIENT-SIDE CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>File Client</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.2/css/all.min.css">
    <style>
        body {
            font-family: 'Poppins', sans-serif;
            background-color: #f4f7fc;
            padding: 20px;
        }
        .container {
            max-width: 600px;
            background: white;
            padding: 20px;
            border-radius: 12px;
            box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
            text-align: center;
        }
        .input-group {
            margin-bottom: 20px;
        }
        .btn-connect {
            width: 100%;
            margin-top: 5%;
        }
        h2 {
            margin-bottom: 20px;
            font-weight: 600;
        }
        .list-group-item {
            transition: 0.3s;
            display: flex;
            justify-content: space-between;
            align-items: center;
            cursor: pointer;
        }
        .list-group-item:hover {
            background-color: #e3f2fd;
        }
        .alert {
            display: none;
        }
        @media (max-width: 768px) {
            .container {
                max-width: 100%;
        }
      }
    </style>
  </head>
  <body>
    <div class="container mt-4">
      <h2>File Client</h2>
      <div class="input-group">
        <input type="text" id="serverIp" class="form-control" placeholder="Enter Server
IP">
        <button class="btn btn-primary btn-connect" onclick="connect()">Connect</button>
      </div>
      <div id="alertBox" class="alert alert-danger"></div>
      <div id="fileList" class="mt-4"></div>
    </div>

    <script>
      let ws = null;

      function connect() {
        const ip = document.getElementById('serverIp').value;
        ws = new WebSocket(`ws://${ip}:8765`);

        ws.onopen = () => {
          console.log('Connected to server');
          showErrorMessage('');
        };

        ws.onerror = () => {
          showErrorMessage('Error: Could not connect to server. Ensure the server is
running.');
        };

        ws.onclose = () => {
          showErrorMessage('Connection closed. Check server status.');
        };

        ws.onmessage = (event) => {
          const msg = JSON.parse(event.data);
          if (msg.type === 'list') {
            showFiles(msg.files);
          } else if (msg.type === 'file') {
            downloadFile(msg.name, msg.content);
          }
        };
      }

      function showFiles(files) {
        const listDiv = document.getElementById('fileList');
        listDiv.innerHTML = '<h3>Available Files:</h3>';
        const listGroup = document.createElement('ul');
        listGroup.className = 'list-group';
        files.forEach(file => {
            const listItem = document.createElement('li');
            listItem.className = 'list-group-item';
            listItem.innerHTML = `<span>${file}</span> <i class="fas fa-download text-
primary" onclick="requestFile('${file}')"></i>`;
            listGroup.appendChild(listItem);
        });

        listDiv.appendChild(listGroup);
    }

    function requestFile(filename) {
        if (ws && ws.readyState === WebSocket.OPEN) {
            ws.send(JSON.stringify({ type: 'download', file: filename }));
        } else {
            showErrorMessage('Error: Not connected to server.');
        }
    }

    function downloadFile(filename, b64Data) {
        const link = document.createElement('a');
        link.download = filename;
        link.href = `data:application/octet-stream;base64,${b64Data}`;
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    }

    function showErrorMessage(message) {
        const alertBox = document.getElementById('alertBox');
        if (message) {
            alertBox.style.display = 'block';
            alertBox.textContent = message;
        } else {
            alertBox.style.display = 'none';
        }
    }
  </script>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

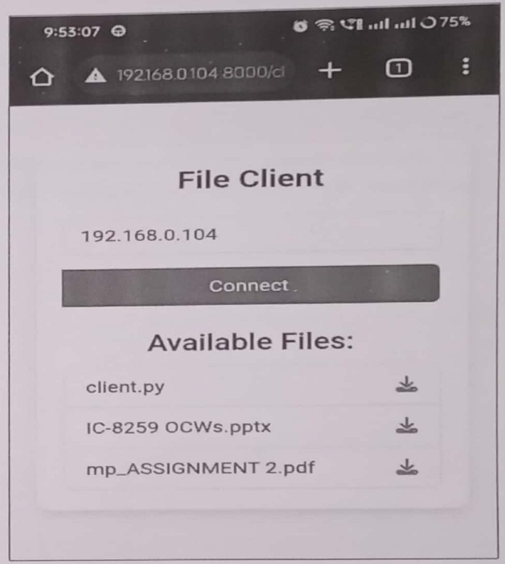
SNAPSHOTS

Server



Server IP: 192.168.0.104
Available Files:
client.py
IC-8259 OCWs.pptx
mp_ASSIGNMENT 2.pdf

Client



File Client

192.168.0.104

Available Files:
client.py
IC-8259 OCWs.pptx
mp_ASSIGNMENT 2.pdf

CONCLUSION

The Seamless File Transfer System successfully demonstrates the integration of Python, Tkinter, WebSockets, and a web-based interface to create an efficient, real-time file-sharing solution. Throughout the development process, we achieved the following:

1. Reliable File Sharing: By leveraging WebSockets, the system enables fast and stable communication between the PC and mobile device, ensuring smooth file transfers over a local Wi-Fi network.
2. User-Friendly Web Interface: The intuitive HTML-based interface allows users to easily browse and download available files without requiring technical expertise.
3. Real-Time Synchronization: The system dynamically updates the available file list, allowing instant access to newly added files.
4. Enhanced Security and Efficiency: Since file transfers occur over a local network, the system eliminates the need for cloud storage, reducing security risks and increasing transfer speeds.

REFERENCES

- **Python Software Foundation. (2021).** Tkinter — Python interface to Tcl/Tk. Retrieved from https://docs.python.org/3/library/tkinter.html
- **NumPy. (2021).** NumPy: The fundamental package for scientific computing with Python. Retrieved from https://numpy.org/
- **WebSockets API. (2021).** Real-Time Bidirectional Communication over WebSockets. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
- **Flask Documentation. (2021).** Flask: A lightweight WSGI web application framework. Retrieved from https://flask.palletsprojects.com/
- **JavaScript Fetch API. (2021).** Handling HTTP Requests in Modern Web Applications. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- **WebRTC Data Channels. (2021).** Peer-to-Peer File Sharing Over Local Networks. Retrieved from https://webrtc.org/getting-started/data-channels