INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO

PRÁCTICA 1:

SISTEMA TIPO DRIVE PARA TRANSFERIR ARCHIVOS

GRUPO:

6CM1

PROFESOR:

AXEL ERNESTO MORENO CERVANTES

ALUMNOS:

FARRERA MENDEZ EMMANUEL SINAI

RAMIREZ LOPEZ FELIPE HIRAM

Introducción

El **envío de archivos** a través de la red es una característica esencial en muchas aplicaciones actuales, como blogs, redes sociales, mensajería instantánea, declaración de impuestos, educación en línea, entre otros. Sin embargo, no todas permiten el envío de archivos de gran tamaño, por ejemplo, el correo electrónico tiene limitaciones de 10 o 20 MB. Esto resalta la necesidad de desarrollar aplicaciones que permitan la transferencia de archivos sin restricciones de tamaño.

Para establecer una conexión entre dos o más dispositivos, se puede utilizar la **arquitectura cliente-servidor**, que se basa en dos roles: el proveedor de recursos (servidor) y el consultor de recursos (cliente). El cliente envía una petición al servidor y espera una respuesta, mientras que el servidor ofrece un servicio que se puede obtener en una red, acepta la petición, realiza el servicio y devuelve el resultado al solicitante.

Esta arquitectura tiene varias ventajas, como su centralización, fácil mantenimiento y escalabilidad, lo que permite aumentar la capacidad de clientes y servidores por separado. Sin embargo, a medida que aumenta el número de clientes, también lo hacen los problemas para el servidor debido a la congestión del tráfico.

En las redes informáticas, un **socket** es un punto final para enviar o recibir datos. Los sockets proporcionan una interfaz común para la comunicación de red en muchos sistemas operativos y se pueden utilizar en muchos lenguajes de programación. Los **sockets de flujo**, un tipo de socket, utilizan protocolos orientados a la conexión, como TCP, lo que garantiza que los mensajes o archivos enviados lleguen en el mismo orden en que se enviaron.

Por lo tanto, la combinación de la arquitectura cliente-servidor con el uso de sockets de flujo puede ser una solución eficaz para el desarrollo de aplicaciones que permitan la transferencia de archivos de cualquier tamaño a través de la red.

Objetivo de la práctica

Implementar un sistema de transferencia de archivos utilizando sockets en Python.

Instrucciones

Desarrollar un cliente y un servidor que se comuniquen entre sí para realizar diversas operaciones, como subir y descargar archivos, eliminar y renombrar archivos y carpetas, copiar archivos, crear nuevos archivos y carpetas, entre otras funciones. Este sistema proporcionará una interfaz gráfica de usuario (GUI) utilizando la biblioteca tkinter para facilitar la interacción del usuario con el sistema de archivos remotos.

Desarrollo

El desarrollo consta de dos partes principales: el cliente y el servidor. La comunicación entre el cliente y el servidor se basa en la definición de comandos que indican la acción a realizar y los detalles asociados (como nombres de archivos o carpetas). Además, el servidor utiliza dos sockets para recibir comandos y transferir archivos, mejorando la eficiencia y permitiendo operaciones concurrentes.

Cliente.py

El cliente está implementado en un script de Python que utiliza la biblioteca tkinter para crear una interfaz gráfica de usuario. La interfaz gráfica muestra dos paneles, cada uno representando una carpeta: una carpeta local y una carpeta remota. El usuario puede realizar diversas acciones, como subir y descargar archivos, eliminar y renombrar archivos y carpetas, copiar archivos, crear nuevos archivos y carpetas, y recargar la ventana de la carpeta local para verificar los cambios hechos previamente. Estas acciones se envían al servidor mediante sockets TCP.

```
import  socket
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
import os
import zipfile

SERVER_HOST = '127.0.0.1'
SERVER_PORT = 12345

RutaL = "local"
RutaR = "remota"

def CrearMenu():
    ventana = tk.Tk()
```

```
    ventana.title("Administrador de Archivos")

    frame_izquierdo = ttk.Frame(ventana)
    frame_izquierdo.pack(side=tk.LEFT, padx=10, pady=10, expand=True,
fill=tk.BOTH)

    frame_derecho = ttk.Frame(ventana)
    frame_derecho.pack(side=tk.RIGHT, padx=10, pady=10, expand=True,
fill=tk.BOTH)
    def listar_directorio(tree, ruta):
        for i in tree.get_children():
            tree.delete(i)

        try:
            contenido = os.listdir(ruta)
            for item in contenido:
                item_path = os.path.join(ruta, item)
                if os.path.isdir(item_path):
                    tree.insert('', 'end', values=(item + "\\",)) #
Agregando un "/" al final para denotar una carpeta
                else:
                    tree.insert('', 'end', values=(item,))
        except FileNotFoundError:
            tree.insert('', 'end', values=("Directorio no encontrado",))

    def SubMenu(sc2, path, accion):
        if accion == "Archivo":
            with open(path, 'wb') as f:
                while True:
                    data = sc2.recv(1024)
                    if not data:
                        break
                    f.write(data)

        elif accion == "Carpeta":
            with open(path + ".zip", 'wb') as f:
                while True:
                    data = sc2.recv(1024)
                    if not data:
                        break
                    f.write(data)
            if not os.path.exists(path):
                os.makedirs(path)
            with zipfile.ZipFile(path + ".zip", 'r') as zipf:
                zipf.extractall(path)
```

```
            os.remove(path + ".zip")

    def boton_presionado(lado, accion, tree):
        print(lado, accion, tree)
        flag = False
        seleccion = tree.selection()

        socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        socket_cliente_2 = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM) socket_cliente.connect((SERVER_HOST,
        SERVER_PORT))
        socket_cliente_2.connect((SERVER_HOST, 12346))

        # Enviar comando al servidor
        if lado == "izquierdo":
            carpeta = "local"
        else:
            carpeta = "remota"

        if seleccion:
            item = tree.item(seleccion[0])
            archivo_seleccionado = item['values'][0]
            print(f"Archivo seleccionado: {archivo_seleccionado}")
            if accion == "Subir":
                comando = f"SUBIR {archivo_seleccionado}"
                flag = True

            elif accion == "Descargar":
                comando = f"DESCARGAR {archivo_seleccionado}"
                flag = True

            elif accion == "Eliminar":
                respuesta = messagebox.askquestion("Confirmar", f"¿Deseas
eliminar '{archivo_seleccionado}'?", icon='warning')
                if respuesta == 'yes':
                    comando = f"ELIMINAR_{carpeta.upper()}
{archivo_seleccionado}"

            elif accion == "Renombrar":
                nuevo_nombre = simpledialog.askstring("Renombrar", f"Nuevo
nombre para '{archivo_seleccionado}' (sin espacios):",
initialvalue=archivo_seleccionado)
                comando = f"RENOMBRAR_{carpeta.upper()}
{archivo_seleccionado} {nuevo_nombre}"
```

```
            elif accion == "Copiar":
                comando = f"COPIAR_{carpeta.upper()} {archivo_seleccionado}"

        else:
            if(accion != "Crear" and accion != "Recargar"):
                messagebox.showwarning("Advertencia", "Seleccionar un
archivo para esta acción")
                accion = "Recargar"
            if accion == "Crear":
                # AQUÍ HAY QUE PONER CÓDIGO PARA QUE UNA VENTANA PREGUNTE SI
QUIERE UN ARCHIVO O UNA CARPETA
                tipo = simpledialog.askstring("Nuevo", f"Nombre para el nuevo
elemento en {lado}:")
                carpeta_nueva = simpledialog.askstring("Crear", f"Nombre para el
nuevo elemento en {lado}:")
                comando = f"CREAR_{tipo.upper()} {carpeta}
            {carpeta_nueva}" elif accion == "Recargar":
                listar_directorio(tree_izquierdo, RutaL)
                listar_directorio(tree_derecho, RutaR)
                comando = ""

            # Envía el comando al servidor
            if comando:
                socket_cliente.send(comando.encode())
                if flag:
                    a_s = archivo_seleccionado.replace("\\",
                    "").replace("/","") if accion == "Subir":
                        path = os.path.join(RutaR, a_s)
                    if "\\" in archivo_seleccionado or "/" in
archivo_seleccionado:
                            SubMenu(socket_cliente_2, path, "Carpeta")
                            messagebox.showinfo("Info", "Carpeta cargada con
éxito")
                    else:
                        SubMenu(socket_cliente_2, path, "Archivo")
                        messagebox.showinfo("Info", "Archivo cargado con
éxito")
                elif accion == "Descargar":
                    path = os.path.join(RutaL, a_s)
                    if "\\" in archivo_seleccionado or "/" in
archivo_seleccionado:
                            SubMenu(socket_cliente_2, path, "Carpeta")
                            messagebox.showinfo("Info", "Carpeta descargada con
éxito")
```

```
                    else:
                        SubMenu(socket_cliente_2, path, "Archivo")
                        messagebox.showinfo("Info", "Archivo descargado con
éxito")
            print(f"Botón '{accion}' presionado en el lado {lado}.")
            listar_directorio(tree_izquierdo, RutaL)
            listar_directorio(tree_derecho, RutaR)
            socket_cliente.close()
            socket_cliente_2.close()
            ventana.update()
        # Carpeta local
        etiqueta_directorio_izquierdo = ttk.Label(frame_izquierdo, text="Carpeta
Local: " + RutaL, width=50)
        etiqueta_directorio_izquierdo.pack(fill=tk.X, pady=5)

        # Botones del lado izquierdo
        botones_izquierdos = ["Subir", "Renombrar", "Eliminar", "Copiar",
"Crear", "Recargar"]
        for btn_text in botones_izquierdos:
            boton = ttk.Button(frame_izquierdo, text=btn_text,
command=lambda accion=btn_text: boton_presionado("izquierdo", accion,
tree_izquierdo)) boton.pack(fill=tk.X, pady=5)

        tree_izquierdo = ttk.Treeview(frame_izquierdo, columns=("LOCAL",),
show="headings")
        tree_izquierdo.heading("LOCAL", text="LOCAL")
        tree_izquierdo.pack(expand=True, fill=tk.BOTH, pady=5)

        # Carpeta remota
        etiqueta_directorio_derecho = ttk.Label(frame_derecho, text="Carpeta
Remota: " + RutaR, width=50)
        etiqueta_directorio_derecho.pack(fill=tk.X, pady=5)

        # Botones del lado derecho
        botones_derechos = ["Descargar", "Renombrar", "Eliminar", "Copiar",
"Crear", "Recargar"]
        for btn_text in botones_derechos:
            boton = ttk.Button(frame_derecho, text=btn_text,
command=lambda accion=btn_text: boton_presionado("derecho", accion,
tree_derecho)) boton.pack(fill=tk.X, pady=5)

        tree_derecho = ttk.Treeview(frame_derecho, columns=("REMOTA",),
show="headings")
        tree_derecho.heading("REMOTA", text="REMOTA")
        tree_derecho.pack(expand=True, fill=tk.BOTH, pady=5)
```

```
    listar_directorio(tree_izquierdo, RutaL)
    listar_directorio(tree_derecho, RutaR)

    ventana.mainloop()
    ventana.update()

def main():
    try:
        # Menu
        CrearMenu()
        socket_salida = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM) _ = socket.socket(socket.AF_INET,
        socket.SOCK_STREAM)
        socket_salida.connect((SERVER_HOST, 12345))
        _.connect((SERVER_HOST, 12346))
        socket_salida.send("SALIR".encode())
        print("Conexión finalizada")
    except Exception as e:
        print(e)

if __name__ == "__main__":
    main()
```

Servidor.py

El servidor está implementado en otro script de Python. Escucha continuamente las conexiones entrantes de clientes en dos puertos diferentes. Utiliza sockets para recibir comandos del cliente y realizar las operaciones correspondientes en el sistema de archivos del servidor. El servidor también maneja la transferencia de archivos y carpetas entre el cliente y el servidor, utilizando compresión ZIP para las carpetas.

```
import socket
import os
import shutil
import zipfile

# Configuración del servidor
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 12345
BUFFER_SIZE = 1024
```

```
LOCAL_FOLDER = "local"
REMOTE_FOLDER = "remota"

# Función para recibir archivos del cliente
def receive_file(client_socket, filename):
    with open(filename, 'wb') as f:
        while True:
            data = client_socket.recv(BUFFER_SIZE)
            if not data:
                break
            f.write(data)

# Función para enviar archivos al cliente
def send_file(client_socket_2, filename):
    with open(filename, 'rb') as f:
        data = f.read(BUFFER_SIZE)
        while data:
            client_socket_2.send(data)
            data = f.read(BUFFER_SIZE)

# Función para recibir una carpeta del cliente
def receive_folder(client_socket, foldername,
    filename): try:
        os.mkdir(foldername) # Crea una carpeta en el servidor con el
nombre especificado
        with open(filename, 'wb') as f:
            while True:
                data = client_socket.recv(BUFFER_SIZE)
                if not data:
                    break
                f.write(data)
    except Exception as e:
        print(f"Error al recibir la carpeta: {str(e)}")


# Función para enviar una carpeta al cliente
def send_folder(client_socket, foldername):
    try:
        # Comprime la carpeta en un archivo ZIP temporal
        shutil.make_archive(foldername, 'zip', foldername)
        with open(foldername + '.zip', 'rb') as f:
            data = f.read(BUFFER_SIZE)
            while data:
                client_socket.send(data)
                data = f.read(BUFFER_SIZE)
```

```
        except Exception as e:
            print(f"Error al enviar la carpeta: {str(e)}")
        finally:
            os.remove(foldername + '.zip')

# Función principal del servidor
def main():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket_2 = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM) server_socket.settimeout(None)
    server_socket_2.settimeout(None)
    server_socket.bind((SERVER_HOST, SERVER_PORT))
    server_socket.listen()
    server_socket_2.bind((SERVER_HOST, 12346))
    server_socket_2.listen()

    print(f"El servidor está escuchando en {SERVER_HOST}:{SERVER_PORT}")

    prendido = True
    while prendido:
        client_socket, client_address = server_socket.accept()
        client_socket_2, client_address_2 =
        server_socket_2.accept() print(f"Se ha conectado un cliente
        desde
{client_address[0]}:{client_address[1]}")

        # Recibe el comando del cliente
        command = client_socket.recv(BUFFER_SIZE).decode()
        print(command)

        if command.startswith('SUBIR'):
            _, filename = command.split()
            filename = filename.replace("\\", "").replace("/", "")
            path = os.path.join(LOCAL_FOLDER, filename)
            if  os.path.exists(path):
                if os.path.isdir(path):
                    send_folder(client_socket_2, path)
                    print(f"Se ha recibido la carpeta '{filename}' de la
carpeta local y se ha guardado en la carpeta remota.")
                else:
                    send_file(client_socket_2, path)
                    print(f"Se ha recibido el archivo '{filename}' de la
carpeta local y se ha guardado en la carpeta remota.")
            else:
                print(f"El archivo o directorio '{filename}' no existe en la
```

```
carpeta local.")


        elif command.startswith('DESCARGAR'):
            _, filename = command.split()
            filename = filename.replace("\\", "").replace("/", "")
            path = os.path.join(REMOTE_FOLDER, filename)

            if  os.path.exists(path):
                if os.path.isdir(path):
                    send_folder(client_socket_2, path)
                    print(f"Se ha enviado la carpeta '{filename}' de la
carpeta remota y de ha guardado en la carpeta local.")
                else:
                    send_file(client_socket_2, path)
                    print(f"Se ha enviado el archivo '{filename}' de la
carpeta remota y de ha guardado en la carpeta local.")
            else:
                print(f"El archivo o directorio '{filename}' no existe en la
carpeta remota.")


        elif command.startswith('ELIMINAR_LOCAL'):
            _, filename = command.split()
            path = os.path.join(LOCAL_FOLDER, filename)
            if os.path.exists(path):
                if os.path.isdir(path):
                    shutil.rmtree(path)
                    print(f"Se ha eliminado el directorio '{filename}' de la
carpeta local.")
                else:
                    os.remove(path)
                    print(f"Se ha eliminado el archivo '{filename}' de la
carpeta local.")
            else:
                print(f"El archivo o directorio '{filename}' no existe en la
carpeta local.")

        elif command.startswith('ELIMINAR_REMOTA'):
            _, filename = command.split()
            path = os.path.join(REMOTE_FOLDER, filename)
            if os.path.exists(path):
                if os.path.isdir(path):
                    shutil.rmtree(path)
                    print(f"Se ha eliminado el directorio '{filename}' de la
```

```
carpeta remota.")
                else:
                    os.remove(path)
                    print(f"Se ha eliminado el archivo '{filename}' de la
carpeta remota.")
            else:
                print(f"El archivo o directorio '{filename}' no existe en la
carpeta remota.")

        elif command.startswith('RENOMBRAR_LOCAL'):
            _, old_name, new_name = command.split()
            os.rename(os.path.join(LOCAL_FOLDER, old_name),
os.path.join(LOCAL_FOLDER, new_name))
            print(f"Se ha renombrado '{old_name}' a '{new_name}' en la
carpeta local.")

        elif command.startswith('RENOMBRAR_REMOTA'):
            _, old_name, new_name = command.split()
            os.rename(os.path.join(REMOTE_FOLDER, old_name),
os.path.join(REMOTE_FOLDER, new_name))
            print(f"Se ha renombrado '{old_name}' a '{new_name}' en la
carpeta remota.")

        elif command.startswith('COPIAR_LOCAL'):
            _, src = command.split()
            src_path = os.path.join(LOCAL_FOLDER, src)
            dest = f"{src}(copia)"
            dest_path = os.path.join(LOCAL_FOLDER, dest)
            if os.path.exists(src_path):
                if os.path.isdir(src_path):
                    shutil.copytree(src_path, dest_path)
                    print(f"Se ha hecho una copia del directorio '{src}' en
'{dest}'.")
                else:
                    shutil.copy2(src_path, dest_path)
                    print(f"Se ha hecho una copia del archivo '{src}' en
'{dest}'.")
            else:
                print(f"El archivo o directorio '{src}' no existe en la
carpeta local.")

        elif command.startswith('COPIAR_REMOTA'):
            _, src = command.split()
            src_path = os.path.join(REMOTE_FOLDER, src)
            dest = f"{src}(copia)"
```

```
            dest_path = os.path.join(REMOTE_FOLDER, dest)
            if os.path.exists(src_path):
                if os.path.isdir(src_path):
                    shutil.copytree(src_path, dest_path)
                    print(f"Se ha hecho una copia del directorio '{src}' en
'{dest}'.")
                else:
                    shutil.copy2(src_path, dest_path)
                    print(f"Se ha hecho una copia del archivo '{src}' en
'{dest}'.")
            else:
                print(f"El archivo o directorio '{src}' no existe en la
carpeta remota.")

        elif command.startswith('CREAR_CARPETA'):
            _, folder, dir_name = command.split()
            if folder == "local":
                os.mkdir(os.path.join(LOCAL_FOLDER, dir_name))
                print(f"Se ha creado el directorio '{dir_name}' en la
carpeta local.")
            elif folder == "remota":
                os.mkdir(os.path.join(REMOTE_FOLDER, dir_name))
                print(f"Se ha creado el directorio '{dir_name}' en la
carpeta remota.")

        elif command.startswith('CREAR_ARCHIVO'):
            _, folder, file_name = command.split()
            file = file_name.split(".", 1)
            if folder == "local":
                open(os.path.join(LOCAL_FOLDER, file_name), 'a').close()
                os.path.join(LOCAL_FOLDER, file_name)
                print(f"Se ha creado el archivo '{file_name}' en la carpeta
local.")
            elif folder == "remota":
                open(os.path.join(REMOTE_FOLDER, file_name), 'a').close()
                os.path.join(REMOTE_FOLDER, file_name)
                print(f"Se ha creado el archivo '{file_name}' en la carpeta
remota.")

        elif command.startswith('SALIR'):
            prendido = False

        client_socket_2.shutdown(socket.SHUT_WR)
        client_socket.close()
```

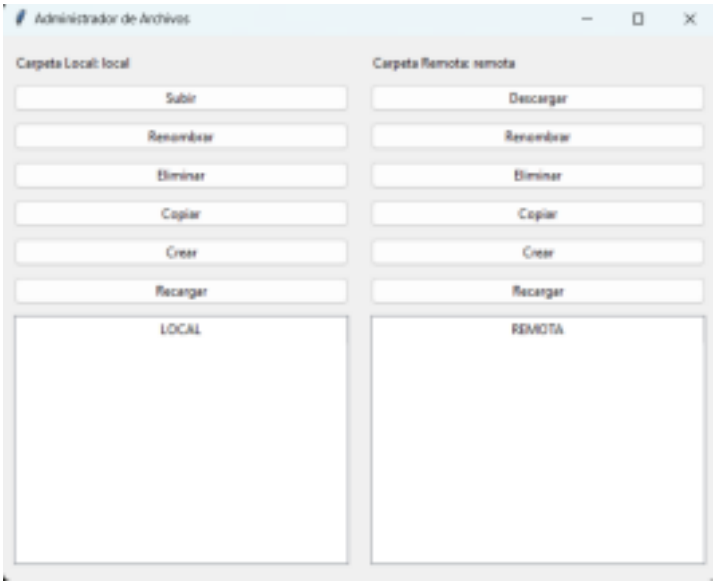
```
if __name__ == "__main__":
    main()
```

```
if not os.path.exists(LOCAL_FOLDER):
    os.mkdir(LOCAL_FOLDER)
if not os.path.exists(REMOTE_FOLDER):
    os.mkdir(REMOTE_FOLDER)
```

Pruebas

Cliente

- El cliente crea una interfaz gráfica de usuario con dos paneles para representar las carpetas local y remota.
- Los botones permiten realizar acciones como subir, descargar, eliminar, renombrar, copiar, crear y recargar.



- Las operaciones seleccionadas por el usuario se envían al servidor a través de sockets TCP.

```
Se ha conectado un cliente desde 127.0.0.1:54187
CREAR_ARCHIVO local hola0.txt
Se ha creado el archivo 'hola0.txt' en la carpeta local.
```

- La interfaz gráfica se actualiza después de realizar cada operación.

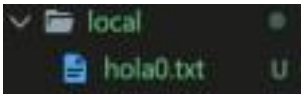
Servidor

- El servidor escucha en dos puertos diferentes, uno para comandos y otro para la transferencia de archivos.

```
El servidor está escuchando en 127.0.0.1:12345
Se ha conectado un cliente desde 127.0.0.1:54176
```

- Recibe comandos del cliente y realiza operaciones en el sistema de archivos del cliente.

```
CREAR_ local
Se ha conectado un cliente desde 127.0.0.1:54187
CREAR_ARCHIVO local hola0.txt
Se ha creado el archivo 'hola0.txt' en la carpeta local.
```



- Gestiona la transferencia de archivos y carpetas entre el cliente y el servidor utilizando sockets y compresión ZIP.
- Proporciona mensajes informativos en la consola sobre las operaciones realizadas.

Conclusión

En esta primera práctica se pudieron probar y aplicar los primeros acercamientos hacia los sockets de flujo, con la implementación del envío de archivos y carpetas entre un cliente y un servidor.

Con ayuda de los ejemplos proporcionados por el profesor se tiene como base el funcionamiento de esta práctica, sin embargo hubo dificultades al realizarla porqueal inicio no teníamos muy claro, como equipo, cómo enviaríamos archivos y carpetas del cliente al servidor, pero ya metiendo las manos poco a poco fuimos acercándonos a la solución.

Consideramos que por las prisas, ya no pudimos mejorar algunos aspectos, sobre todo no funcionales, del sistema, como lo son la elección del tipo de “objeto” a crear (archivo o carpeta), que no es tan intuitivo como esperábamos. Sin embargo, esto no afecta el buen funcionamiento del sistema.

- **Farrera Mendez Emmanuel Sinai**

En esta primera práctica, exploramos y aplicamos los conceptos básicos de los sockets de flujo al implementar la transferencia de archivos y carpetas entre un cliente y un servidor.

Inicialmente, nos enfrentamos a algunos desafíos al comprender cómo enviar archivos y carpetas del cliente al servidor. Sin embargo, con la orientación del profesor y mediante el trabajo en equipo, pudimos superar estas dificultades y avanzar hacia la solución.

A pesar de las limitaciones de tiempo, logramos implementar la funcionalidad principal del sistema. Sin embargo, reconocemos que quedan aspectos por mejorar, especialmente en la experiencia del usuario y la intuitividad del sistema para elegir entre crear un archivo o una carpeta. Aunque hay áreas que podrían mejorarse, estamos satisfechos con el funcionamiento general del sistema y estamos comprometidos a seguir refinando y mejorando nuestra solución en futuras iteraciones.

- **Ramírez López Felipe Hiram**