

Practica 1

IMPLEMENTACIÓN DE APLICACIÓN ESTILO ONEDRIVE

MATERIA

Aplicaciones para comunicaciones de red

PROFESOR

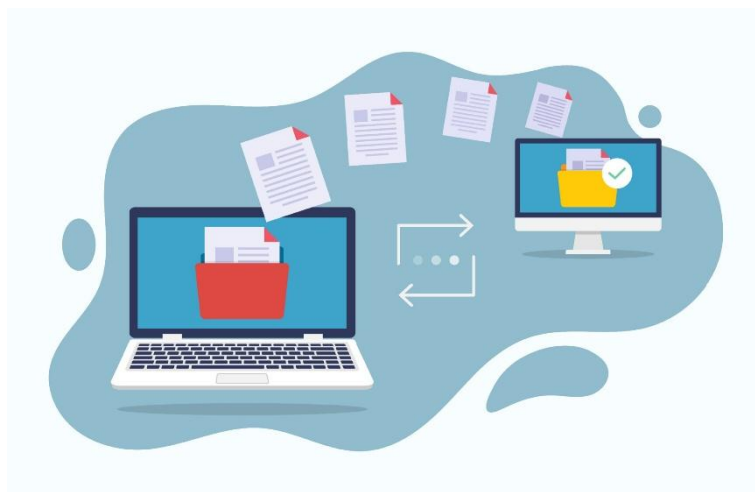
Moreno Cervantes Axel Ernesto

GRUPO

6CM2

ALUMNOS

- Guevara Badillo Areli Alejandra
- Ramírez Martínez Alejandro



FECHA

lunes, 14 de abril de 2025

INTRODUCCIÓN

En esta práctica se desarrolló una aplicación con arquitectura cliente-servidor que simula el funcionamiento básico de una plataforma de almacenamiento en la nube, similar a OneDrive. El sistema permite al usuario gestionar archivos y carpetas de manera remota mediante una interfaz gráfica amigable. Para lograrlo, se implementó un servidor capaz de recibir y ejecutar instrucciones desde uno o varios clientes, permitiendo así operaciones como transferencia de archivos, creación y eliminación de elementos, y visualización del contenido de directorios remotos.

La comunicación entre el cliente y el servidor se lleva a cabo a través de sockets, utilizando un protocolo simple basado en el envío de códigos de operación para identificar las distintas acciones solicitadas. Esta implementación permite reforzar conceptos clave como la programación en red, la manipulación de archivos en Java, y el diseño de interfaces gráficas.

OBJETIVOS

- Desarrollar una aplicación distribuida que permita la gestión remota de archivos y carpetas mediante una interfaz gráfica, utilizando comunicación por sockets entre cliente y servidor.
- Implementar una interfaz gráfica que permita seleccionar carpetas locales y remotas.
- Visualizar el contenido de ambas carpetas dentro de la aplicación.
- Permitir la transferencia de archivos y carpetas entre la máquina local y el servidor.
- Implementar funcionalidades para crear, renombrar y eliminar archivos y carpetas en el servidor.
- Establecer una comunicación eficiente entre cliente y servidor usando sockets TCP.
- Aplicar principios de programación orientada a objetos y manejo de archivos en Java.

DESARROLLO

Esta práctica implementa una conexión confiable entre una carpeta local y una carpeta remota utilizando **sockets de flujo**. La práctica está conformada por tres componentes principales:

- **DropBox.java:** Esta clase representa la interfaz gráfica de la aplicación del lado del cliente. Está construida utilizando Swing y contiene los elementos visuales con los cuales el usuario interactúa para realizar las operaciones. Este archivo es responsable de la experiencia del usuario, pero no maneja directamente los archivos ni la red; esas tareas las ejecuta la clase Cliente.
- **Cliente.java:** Este archivo contiene toda la lógica de operaciones del lado cliente. Se encarga de manejar archivos locales, enviar peticiones al servidor y procesar las respuestas. Utiliza sockets (Socket, DataInputStream, DataOutputStream) para comunicarse con el servidor. El cliente trabaja en coordinación con DropBox.java, respondiendo a las acciones del usuario. Cada operación que requiere trabajar con la carpeta remota genera una conexión temporal con el servidor, envía un código que indica la acción deseada, y transmite o recibe datos según sea necesario.
- **Servidor.java:** Esta clase implementa el servidor que atiende a las peticiones de los clientes. Está siempre a la escucha en un puerto (4444) y atiende múltiples peticiones utilizando un hilo por cada conexión (Thread). El servidor actúa como el administrador de archivos del lado remoto, ejecutando cada petición que recibe en una nueva instancia del hilo Servidor, lo cual le permite atender varios clientes al mismo tiempo o varias operaciones consecutivas sin bloquearse.

Estos tres componentes en torno a **una aplicación de sincronización de archivos** entre un cliente y un servidor, al estilo de un sistema de almacenamiento en la nube, similar a lo que hacen aplicaciones como OneDrive, pero de forma simplificada.

DROPBOX.JAVA (INTERFAZ GRÁFICA)

La clase DropBox tiene la función de ser la interfaz gráfica con la que interactúa el usuario. A través de esta interfaz, el usuario puede seleccionar carpetas locales y remotas, realizar transferencias de archivos y ver el progreso de estas.

Componentes de la interfaz gráfica:

Se crean los componentes gráficos que se van a usar.

- **modeloLocal** y **modeloRemoto** son modelos de lista que contienen los archivos y carpetas tanto locales como remotas.
- **listaLocal** y **listaRemota** son las listas de la interfaz que muestran estos archivos.
- **barraProgreso** es una barra de progreso que se actualizará durante las transferencias de archivos.

```
public static DefaultListModel<String> modeloLocal = new DefaultListModel<>();
public static DefaultListModel<String> modeloRemoto = new DefaultListModel<>();
public static JList<String> listaLocal = new JList<>(modeloLocal);
public static JList<String> listaRemota = new JList<>(modeloRemoto);
public static JProgressBar barraProgreso = new JProgressBar();
```

Seleccionar Carpeta Local:

Esta función abre un cuadro de diálogo para que el usuario seleccione una carpeta local. Una vez seleccionada, se guarda la ruta y se actualiza la vista local.

```
public static void seleccionarCarpetaLocal() {
    JFileChooser chooser = new JFileChooser();
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    int r = chooser.showOpenDialog(null);
    if (r == JFileChooser.APPROVE_OPTION) {
        carpetaLocal = chooser.getSelectedFile().getAbsolutePath();
        rutaActualLocal = carpetaLocal;
        actualizarVistaLocal();
    }
}
```

Actualizar las vistas de las carpetas:

Esta función se encarga de actualizar la lista de archivos y carpetas locales. Si la ruta local no es nula o vacía, se obtienen los archivos de la carpeta y se agregan a la lista.

- Si es una carpeta, se añade un ícono (\u21b3).
- Si es un archivo, simplemente se muestra su nombre.

```
public static void actualizarVistaLocal() {
    DropBox.modeloLocal.clear();
    if (rutaActualLocal == null || rutaActualLocal.isEmpty()) return;
    File f = new File(rutaActualLocal);
    File[] archivos = f.listFiles();
    if (archivos != null) {
        Arrays.sort(archivos);
    }
}
```

```

        for (File archivo : archivos) {
            if (archivo.isDirectory())
                DropBox.modeloLocal.addElement("\u21b3 " + archivo.getName());
            else
                DropBox.modeloLocal.addElement(archivo.getName());
        }
    }
}

```

Transferir archivos:

Esta función se encarga de enviar un archivo al servidor.

- Si el archivo es un directorio, la función lo procesa recursivamente.
- Para los archivos, se crea un socket y se establece la conexión con el servidor.
- Se envía el archivo en fragmentos (por bloques de 2000 bytes), y se actualiza la barra de progreso mientras se envían los datos.

```

private static void enviarArchivo(File f) {
    if (f.isDirectory()) {
        for (File sub : f.listFiles()) {
            enviarArchivo(sub);
        } return;
    }
    try (Socket cl = new Socket(host, pto)) {
        DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
        FileInputStream fis = new FileInputStream(f);
        String rutaRelativa = f.getAbsolutePath().replace(carpetalocal + sep, "");
        dos.writeInt(0); // Código 0: subir archivo
        dos.writeUTF(rutaRelativa);
        dos.writeLong(f.length());
        long enviados = 0;
        int n;
        byte[] b = new byte[2000];
        while ((n = fis.read(b)) != -1) {
            dos.write(b, 0, n);
            enviados += n;
            int porcentaje = (int) ((enviados * 100) / f.length());
            DropBox.barraProgreso.setValue(porcentaje);
        }
        fis.close();
        System.out.println("Archivo enviado: " + rutaRelativa);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

CLIENTE.JAVA

La clase Cliente maneja la lógica principal del lado del cliente. Su función es la de realizar las operaciones de red, como la transferencia de archivos entre el cliente y el servidor.

Conexión con el servidor:

Esta función se encarga de enviar la ruta remota seleccionada al servidor. El código 10 indica que se está cambiando la ruta en el servidor.

```
private static void enviarRutaRemota(String ruta) {
    try (Socket c1 = new Socket(host, pto)) {
        DataOutputStream dos = new DataOutputStream(c1.getOutputStream());
        dos.writeInt(10);
        dos.writeUTF(ruta);
        dos.flush();
    } catch (Exception e) {
        e.printStackTrace();
    }
    actualizarVistaRemota();
}
```

Actualizar la vista remota:

Esta función se conecta al servidor y solicita la lista de archivos y carpetas remotas. Los resultados se añaden al modelo de lista remota (modeloRemoto).

```
public static void actualizarVistaRemota() {
    try (Socket c1 = new Socket(host, pto)) {
        DataOutputStream dos = new DataOutputStream(c1.getOutputStream());
        dos.writeInt(1);
        dos.flush();
        DataInputStream dis = new DataInputStream(c1.getInputStream());
        int n = dis.readInt();
        DropBox.modeloRemoto.clear();
        for (int i = 0; i < n; i++) {
            String nombre = dis.readUTF();
            boolean esCarpeta = dis.readBoolean();
            if (esCarpeta) {
                DropBox.modeloRemoto.addElement("\u21b3 " + nombre);
            } else {
                DropBox.modeloRemoto.addElement(nombre);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Recibir y transferir archivos:

Esta función gestiona las transferencias de archivos desde y hacia el servidor. Si se transfieren archivos locales al servidor, se utiliza la función `enviarArchivo()`. Si el cliente solicita recibir archivos del servidor, la función gestiona el proceso de descarga, similar a cómo funciona la subida de archivos.

```
public static void transferirArchivos(boolean enRemoto) {
    if (enRemoto) {
        int[] indices = DropBox.listaLocal.getSelectedIndices();
        if (indices.length == 0) {
            JOptionPane.showMessageDialog(null, "Seleccione archivos locales para transferir.");
            return;
        }
        for (int i : indices) {
            String nombre = DropBox.modeloLocal.getElementAt(i);
            File f = new File(rutaActualLocal + sep + nombre.replace("\u21b3 ", ""));
            enviarArchivo(f);
        }
        actualizarVistaRemota();
    } else {
        int[] indices = DropBox.listaRemota.getSelectedIndices();
        if (indices.length == 0) {
            JOptionPane.showMessageDialog(null, "Seleccione archivos remotos para transferir.");
            return;
        }
        try (Socket cl = new Socket(host, pto)) {
            DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
            DataInputStream dis = new DataInputStream(cl.getInputStream());
            for (int i : indices) {
                String nombreOriginal = DropBox.modeloRemota.getElementAt(i);
                boolean esCarpeta = nombreOriginal.startsWith("\u21b3");
                String nombre = nombreOriginal.replace("\u21b3 ", "");
                if (esCarpeta) {
                    dos.writeInt(11);
                    dos.writeUTF(nombre);
                    int cantidad = dis.readInt();
                    for (int j = 0; j < cantidad; j++) {
                        String rutaRel = dis.readUTF();
                        long tam = dis.readLong();
                        File destino = new File(rutaActualLocal + sep + rutaRel);
                        destino.getParentFile().mkdirs();
                        FileOutputStream fos = new FileOutputStream(destino);
                        long recibidos = 0;
                        int n;
```

```

        byte[] b = new byte[2000];
        while (recibidos < tam) {
            n = dis.read(b);
            fos.write(b, 0, n);
            recibidos += n;
            int porcentaje = (int) ((recibidos * 100) / tam);
            DropBox.barraProgreso.setValue(porcentaje);
        }
        fos.close();
        System.out.println("Archivo recibido: " + rutaRel);
    }
} else {
    dos.writeInt(2);
    dos.writeInt(1);
    dos.writeUTF(nombre);
    String nombreArchivo = dis.readUTF();
    long tam = dis.readLong();
    File out = new File(rutaActualLocal + sep + nombreArchivo);
    FileOutputStream fos = new FileOutputStream(out);
    long recibidos = 0;
    int n;
    byte[] b = new byte[2000];
    while (recibidos < tam) {
        n = dis.read(b);
        fos.write(b, 0, n);
        recibidos += n;
        int porcentaje = (int) ((recibidos * 100) / tam);
        DropBox.barraProgreso.setValue(porcentaje);
    }
    fos.close();
    System.out.println("Archivo recibido: " + nombreArchivo);
}
}
} catch (Exception e) {
    e.printStackTrace();
}
}
actualizarVistaLocal();
}
}

```

SERVIDOR.JAVA

El servidor está encargado de recibir las peticiones de los clientes y ejecutar las operaciones solicitadas, como la transferencia de archivos, eliminación, creación, etc.

Conexión y procesamiento de peticiones:

Este bloque es el principal del servidor, donde se lee la operación solicitada por el cliente (mediante el código de operación) y se ejecuta la correspondiente, como subir un archivo, obtener la lista de archivos, etc.

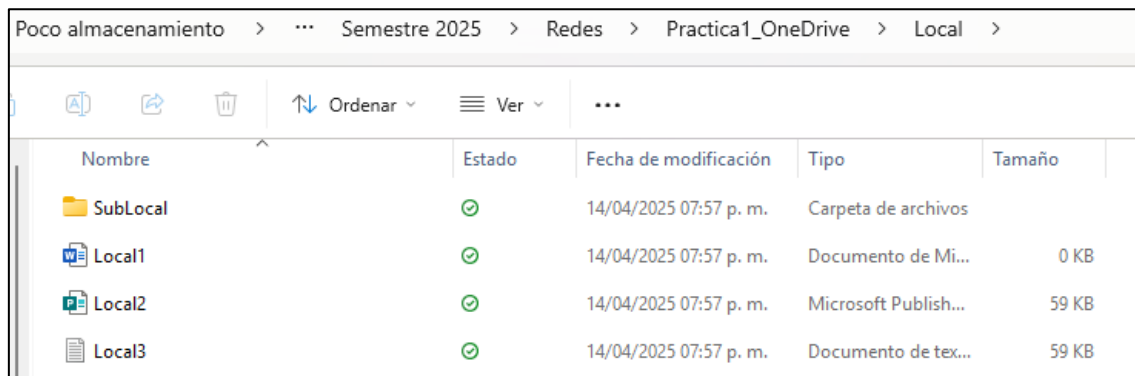
```
public void run() {
    try {
        DataInputStream dis = new DataInputStream(socket.getInputStream());
        int operacion = dis.readInt();
        switch (operacion) {
            case 0:
                recibirArchivo();
                break;
            case 1:
                enviarListaArchivos();
                break;
            case 2:
                recibirArchivoDescarga();
                break;
            case 5:
                eliminarArchivos();
                break;
            case 6:
                crearArchivo();
                break;
            case 7:
                crearCarpeta();
                break;
            case 8:
                renombrarArchivo();
                break;
            case 10:
                cambiarRuta();
                break;
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```


PRUEBAS Y RESULTADOS

Para verificar el correcto funcionamiento de la aplicación, se realizaron pruebas en un entorno simulado, se crearon dos carpetas, una local y una remota, estas están comunicadas mediante una interfaz cliente-servidor, lo cuál nos permite realizar operaciones como copiar una carpeta/archivo desde la carpeta local hacia la carpeta remota y viceversa, además de poder cambiar el nombre, crear o eliminar estos archivos y carpetas.

Carpeta Local sin modificaciones:

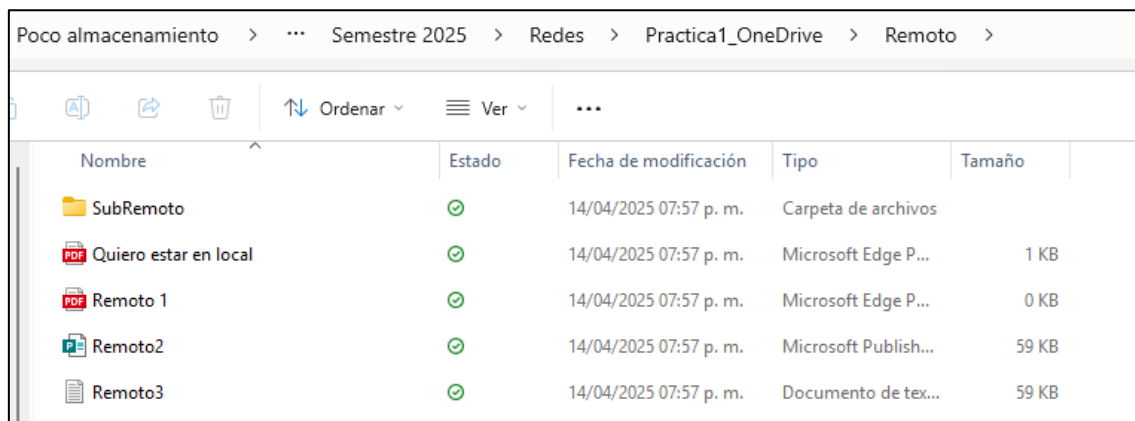
Se encuentra alojada en nuestro proyecto, hemos creado algunas subcarpetas y archivos para la visualización del funcionamiento de la práctica.



Nombre	Estado	Fecha de modificación	Tipo	Tamaño
SubLocal	✓	14/04/2025 07:57 p. m.	Carpeta de archivos	
Local1	✓	14/04/2025 07:57 p. m.	Documento de Mi...	0 KB
Local2	✓	14/04/2025 07:57 p. m.	Microsoft Publish...	59 KB
Local3	✓	14/04/2025 07:57 p. m.	Documento de tex...	59 KB

Carpeta Remota sin modificaciones:

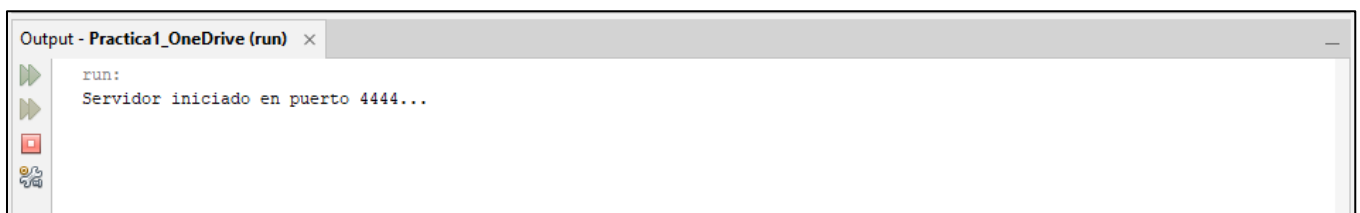
Se encuentra alojada en nuestro proyecto, hemos creado algunas subcarpetas y archivos para la visualización del funcionamiento de la práctica.



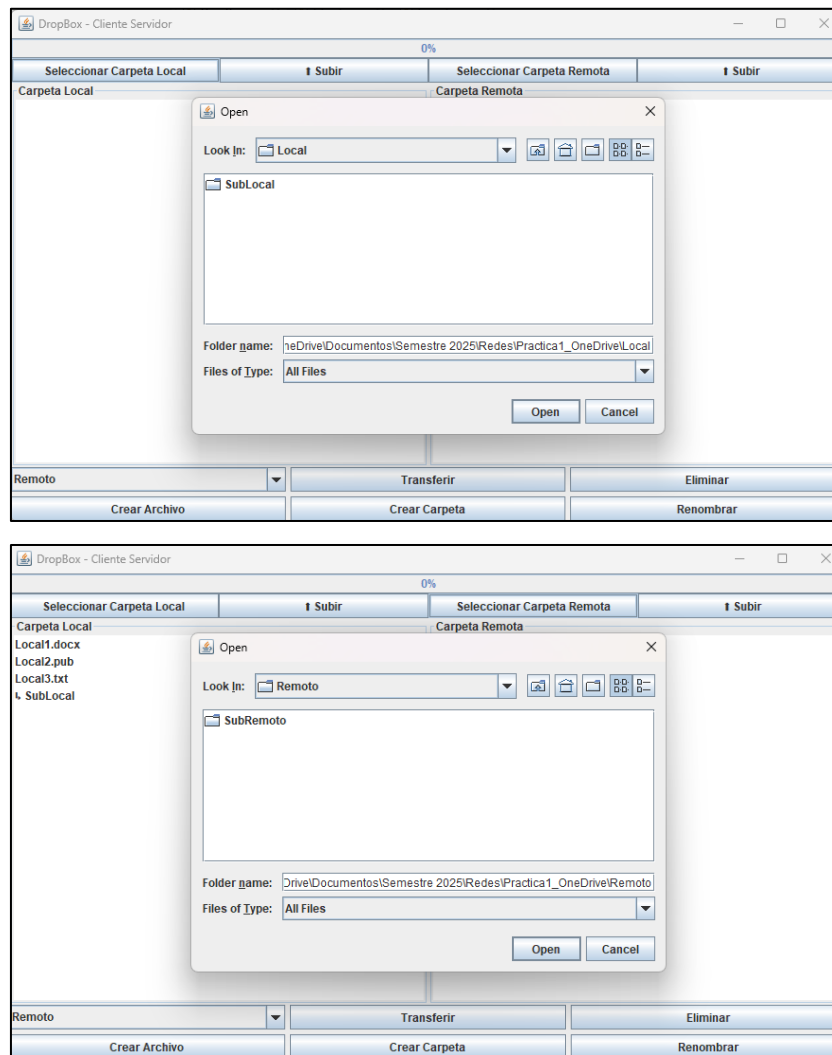
Nombre	Estado	Fecha de modificación	Tipo	Tamaño
SubRemoto	✓	14/04/2025 07:57 p. m.	Carpeta de archivos	
Quiero estar en local	✓	14/04/2025 07:57 p. m.	Microsoft Edge P...	1 KB
Remoto 1	✓	14/04/2025 07:57 p. m.	Microsoft Edge P...	0 KB
Remoto2	✓	14/04/2025 07:57 p. m.	Microsoft Publish...	59 KB
Remoto3	✓	14/04/2025 07:57 p. m.	Documento de tex...	59 KB

Ejecución del DropBox:

Este archivo debe ejecutarse primero, pues es el responsable de echar a andar el Servidor y el Cliente.

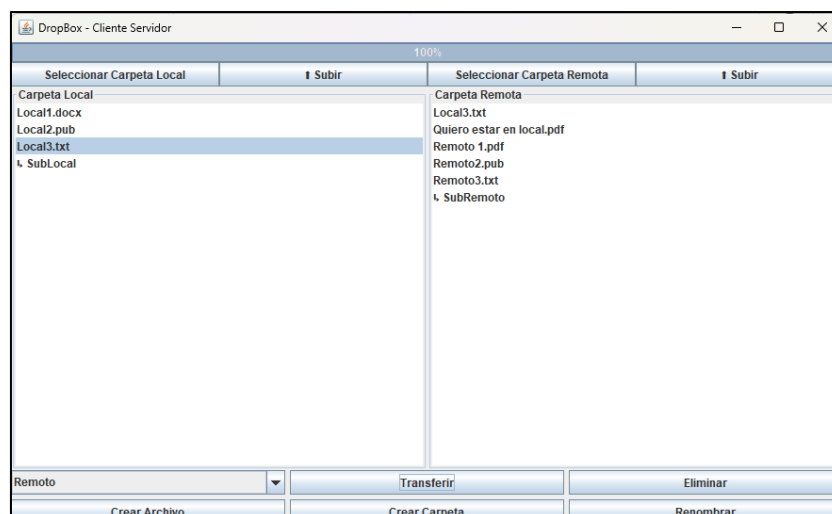


Debemos seleccionar la carpeta Local y Remota según corresponda.



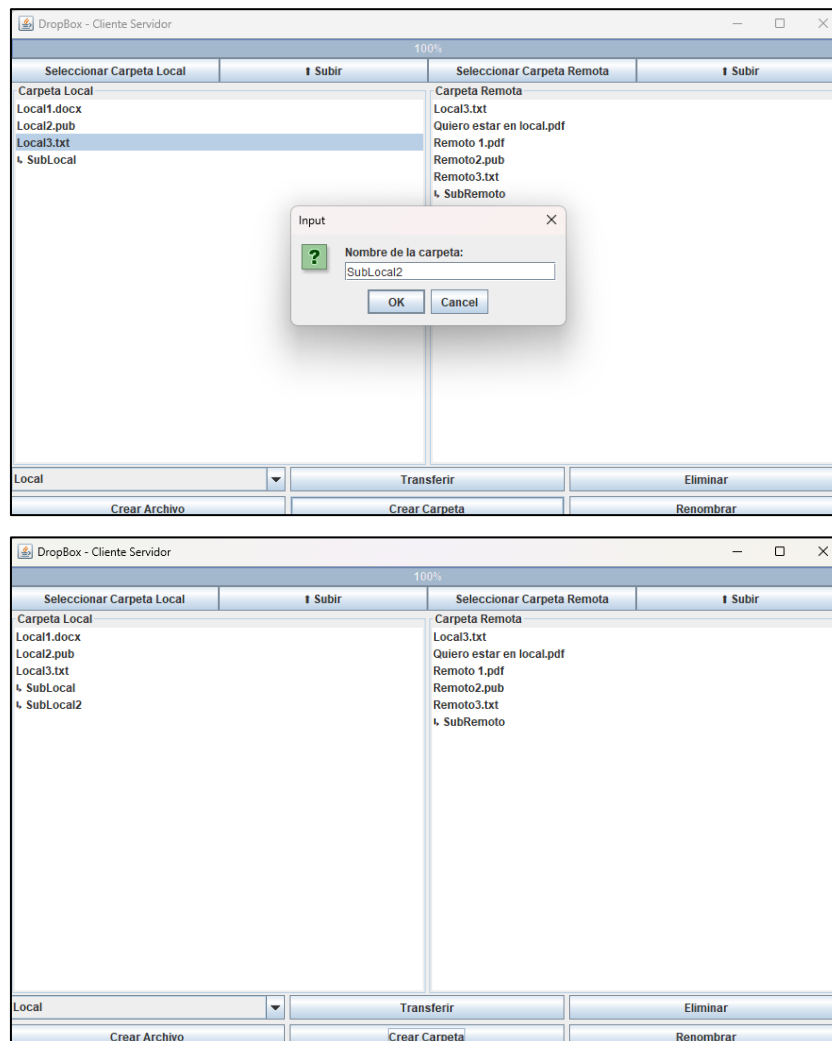
Transferir archivos o carpetas:

Debemos seleccionar la carpeta en la que queremos guardar el archivo (ejemplo Remoto) y seleccionar este del contrario (ejemplo Local).



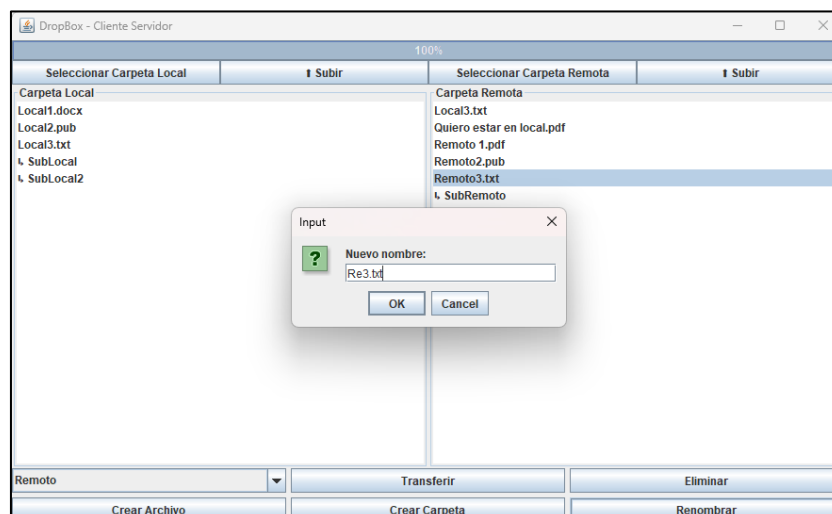
Crear archivos o carpetas:

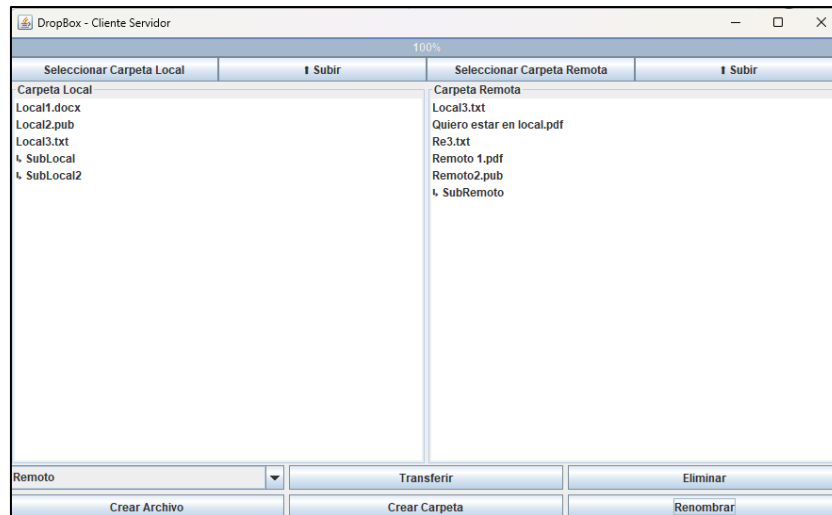
Debemos seleccionar la carpeta donde crearemos la carpeta (ejemplo Local) y escribir el nombre que le queremos dar.



Renombrar archivos o carpetas:

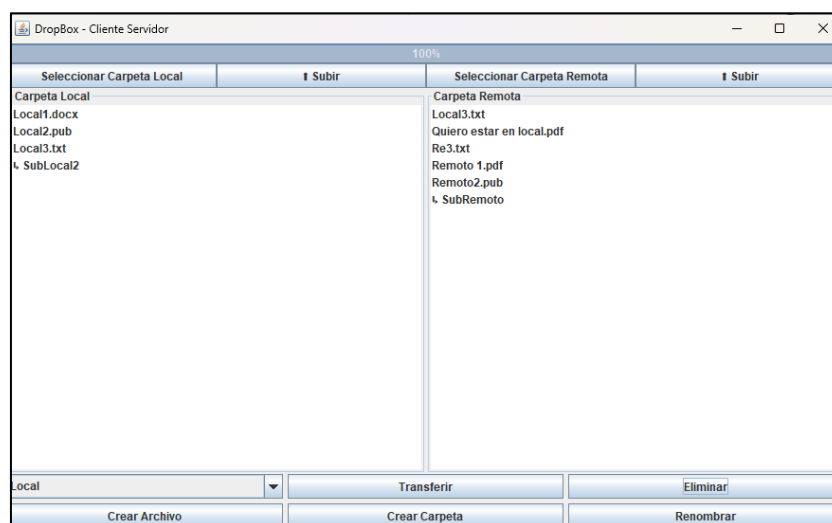
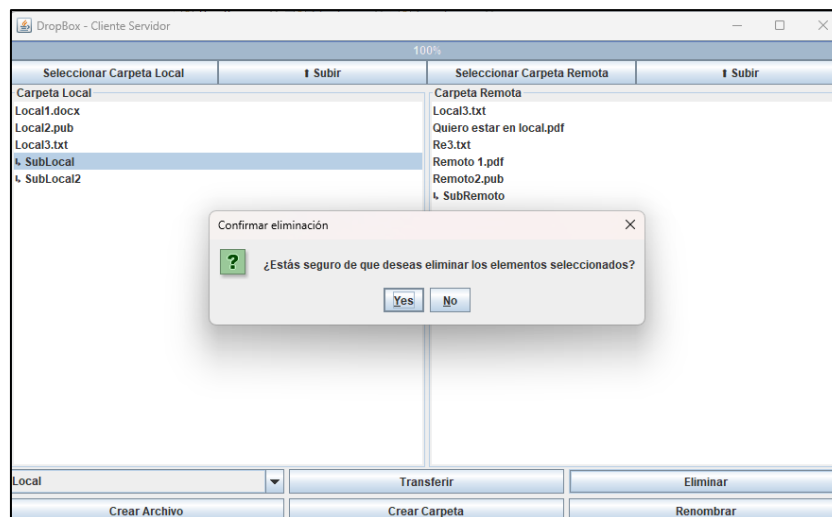
Elegimos la carpeta (Local/Remota) y el archivo a cambiar el nombre, luego seleccionamos Renombrar y podremos hacer este cambio.





Eliminar archivos o carpetas:

Hacemos algo muy similar a lo anterior hecho, primero elegimos la carpeta (Local/Remota), seleccionamos la carpeta a borrar y listo.



Si cerramos la ventana, podremos ver que se ha cerrado el servidor y todo el programa.

Output - Practica1_OneDrive (run) x

```
run:
Servidor iniciado en puerto 4444...
Cliente conectado desde /127.0.0.1:53747
Ruta remota ahora apunta a: C:\Users\erick\OneDrive\Documentos\Semestre 2025\Redes\Practical_OneDrive\Remoto
Cliente conectado desde /127.0.0.1:53748
Lista enviada con 5 elementos.
Cliente conectado desde /127.0.0.1:53749
Archivo enviado: Local3.txt
Archivo recibido: Local3.txt (59904 bytes)
Cliente conectado desde /127.0.0.1:53750
Lista enviada con 6 elementos.
Cliente conectado desde /127.0.0.1:53866
Renombrado: Remoto3.txt -> Re3.txt
Cliente conectado desde /127.0.0.1:53867
Lista enviada con 6 elementos.
Servidor finalizado.
BUILD SUCCESSFUL (total time: 15 minutes 31 seconds)
```

Revisión de las carpetas:

Podemos percatarnos que se han realizado los cambios en nuestras carpetas Local y Remota.

OneDrive	>	...	Documentos	>	Semestre 2025	>	Redes	>	Practica1_OneDrive	>	Remoto	>
Ordenar Ver ...												
Nombre			Estado		Fecha de modificación		Tipo		Tamaño			
SubRemoto			✓		14/04/2025 07:57 p. m.		Carpeta de archivos					
Local3			✓		14/04/2025 10:13 p. m.		Documento de tex...		59 KB			
Quiero estar en local			✓		14/04/2025 07:57 p. m.		Microsoft Edge P...		1 KB			
Re3			✓		14/04/2025 07:57 p. m.		Documento de tex...		59 KB			
Remoto 1			✓		14/04/2025 07:57 p. m.		Microsoft Edge P...		0 KB			
Remoto2			✓		14/04/2025 07:57 p. m.		Microsoft Publish...		59 KB			

OneDrive	>	...	Documentos	>	Semestre 2025	>	Redes	>	Practica1_OneDrive	>	Local	>
Ordenar Ver ...												
Nombre			Estado		Fecha de modificación		Tipo		Tamaño			
SubLocal2			✓		14/04/2025 10:15 p. m.		Carpeta de archivos					
Local1			✓		14/04/2025 07:57 p. m.		Documento de Mi...		0 KB			
Local2			✓		14/04/2025 07:57 p. m.		Microsoft Publish...		59 KB			
Local3			✓		14/04/2025 07:57 p. m.		Documento de tex...		59 KB			

Los resultados de las pruebas demostraron que:

- La aplicación permite seleccionar correctamente carpetas locales y remotas desde una interfaz gráfica, estableciendo rutas de trabajo para ambas partes.
- Se logró listar adecuadamente el contenido de las carpetas, incluyendo archivos y subcarpetas, tanto en el cliente como en el servidor.
- Las operaciones de creación, eliminación y renombramiento de archivos y carpetas funcionan correctamente en ambas ubicaciones (local y remota).
- Los archivos y carpetas se transfieren exitosamente desde el cliente al servidor y viceversa, incluso cuando se trata de directorios completos con múltiples archivos internos.
- El uso de dos sockets (uno para control y otro para datos) permitió separar de manera eficiente las instrucciones de manejo y la transferencia de contenido, tal como lo establece el protocolo FTP.
- La barra de progreso integrada reflejó adecuadamente el estado de las transferencias, brindando retroalimentación visual al usuario.

CONCLUSIONES

La implementación de esta aplicación estilo OneDrive permitió comprender y aplicar los principios básicos del protocolo FTP definidos en la RFC 959, especialmente en lo relacionado con el uso de primitivas de control para gestionar comandos. Se logró desarrollar una arquitectura cliente-servidor que utiliza dos sockets de flujo: uno para el canal de control y otro para la transferencia de datos, simulando así la estructura del protocolo FTP tradicional. A través de una interfaz gráfica intuitiva, se ofreció al usuario la capacidad de seleccionar carpetas locales y remotas, visualizar su contenido, y realizar operaciones fundamentales como creación, eliminación, renombrado y transferencia de archivos o carpetas en ambos sentidos. La práctica no solo reforzó conceptos de redes y comunicación entre procesos, sino también el diseño de aplicaciones distribuidas robustas y funcionales.