Universidad Rafael Landívar

Facultad de Ingeniería

Carrera: Licenciatura en Ingeniería en Informática y Sistemas

Curso: Manejo e Implementación de Archivos

Ingeniero: Ángel Ricardo Trujillo Mazariegos

Proyecto No.2

“Simulador FAT con Interfaz Gráfica en Python”

Nombre: Anthony Rafael Domínguez Arriola

Diego Alejandro Afre Reyes

Carné: 163952 - 1594422

Código Fuente:

```
import os
import json
import uuid
from datetime import datetime
import tkinter as tk
from tkinter import messagebox, simpledialog, filedialog
DATA_DIR = "fat_data"
BLOCKS_DIR = os.path.join(DATA_DIR, "blocks")
FAT_PATH = os.path.join(DATA_DIR, "fat_table.json")
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
if not os.path.exists(BLOCKS_DIR):
    os.makedirs(BLOCKS_DIR)
class FATManager:
    def __init__(self, path):
        self.path = path
        self.fat = {}
        self.load()
    def load(self):
        if os.path.exists(self.path):
            with open(self.path, "r", encoding="utf-8") as f:
                try:
                    self.fat = json.load(f)
```

```
            except Exception:
                self.fat = {}
        else:
            self.fat = {}
            self.save()
    def save(self):
        with open(self.path, "w", encoding="utf-8") as f:
            json.dump(self.fat, f, indent=2, ensure_ascii=False)
    def create_file(self, name, content, owner):
        file_id = str(uuid.uuid4())
        blocks = self._create_blocks(file_id, content)
        now = datetime.now().isoformat()
        entry = {
            "id": file_id,
            "name": name,
            "data_path": blocks[0] if blocks else "",
            "trash": False,
            "size": len(content),
            "created": now,
            "modified": now,
            "deleted": None,
            "owner": owner,
            "permissions": {owner: {"read": True, "write": True}}
        }
```

```
        self.fat[file_id] = entry
        self.save()
        return file_id
    def _create_blocks(self, file_id, content):
        if content == "":
            block_path = os.path.join(BLOCKS_DIR, f"{file_id}_0.json")
            block = {"datos": "", "siguiente": None, "eof": True}
            with open(block_path, "w", encoding="utf-8") as f:
                json.dump(block, f, ensure_ascii=False)
            return [block_path]
        parts = [content[i:i+20] for i in range(0, len(content), 20)]
        paths = []
        for i, part in enumerate(parts):
            path = os.path.join(BLOCKS_DIR, f"{file_id}_{i}.json")
            paths.append(path)
        for i, part in enumerate(parts):
            next_path = paths[i+1] if i+1 < len(paths) else None
            block = {"datos": part, "siguiente": next_path, "eof": next_path is None}
            with open(paths[i], "w", encoding="utf-8") as f:
                json.dump(block, f, ensure_ascii=False)
        return paths
    def read_file(self, file_id, user):
        entry = self.fat.get(file_id)
        if not entry:
```

```
            raise FileNotFoundError
        perms = entry.get("permissions", {})
        if not (perms.get(user, {}).get("read") or entry.get("owner") == user):
            raise PermissionError
        path = entry.get("data_path")
        content = ""
        p = path
        while p:
            if not os.path.exists(p):
                break
            with open(p, "r", encoding="utf-8") as f:
                b = json.load(f)
            content += b.get("datos", "")
            p = b.get("siguiente")
        return entry, content
    def modify_file(self, file_id, new_content, user):
        entry = self.fat.get(file_id)
        if not entry:
            raise FileNotFoundError
        perms = entry.get("permissions", {})
        if not (perms.get(user, {}).get("write") or entry.get("owner") == user):
            raise PermissionError
        old_blocks = self._collect_block_paths(entry.get("data_path"))
        new_blocks = self._create_blocks(file_id, new_content)
```

```
        entry["data_path"] = new_blocks[0] if new_blocks else ""
        entry["size"] = len(new_content)
        entry["modified"] = datetime.now().isoformat()
        self.fat[file_id] = entry
        self.save()
        for p in old_blocks:
            try:
                if os.path.exists(p):
                    os.remove(p)
            except Exception:
                pass
    def _collect_block_paths(self, start_path):
        paths = []
        p = start_path
        while p:
            if not os.path.exists(p):
                break
            paths.append(p)
            with open(p, "r", encoding="utf-8") as f:
                b = json.load(f)
            p = b.get("siguiente")
        return paths
    def delete_file(self, file_id, user):
        entry = self.fat.get(file_id)
```

```
        if not entry:
            raise FileNotFoundError
        entry["trash"] = True
        entry["deleted"] = datetime.now().isoformat()
        self.fat[file_id] = entry
        self.save()
    def recover_file(self, file_id, user):
        entry = self.fat.get(file_id)
        if not entry:
            raise FileNotFoundError
        entry["trash"] = False
        entry["deleted"] = None
        entry["modified"] = datetime.now().isoformat()
        self.fat[file_id] = entry
        self.save()
    def list_files(self, include_trash=False):
        res = []
        for fid, e in self.fat.items():
            if include_trash:
                res.append(e)
            else:
                if not e.get("trash"):
                    res.append(e)
        return sorted(res, key=lambda x: x.get("name"))
```

```
    def set_permission(self, file_id, owner_user, target_user, read, write):
        entry = self.fat.get(file_id)
        if not entry:
            raise FileNotFoundError
        if entry.get("owner") != owner_user:
            raise PermissionError
        perms = entry.get("permissions", {})
        perms[target_user] = {"read": read, "write": write}
        entry["permissions"] = perms
        entry["modified"] = datetime.now().isoformat()
        self.fat[file_id] = entry
        self.save()
    def get_entry(self, file_id):
        return self.fat.get(file_id)
fat = FATManager(FAT_PATH)
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Simulador FAT - Proyecto")
        self.current_user = tk.StringVar(value="admin")
        self.available_users = ["admin", "alice", "bob"]
        self.setup_ui()
        self.refresh_list()
    def setup_ui(self):
```

```
        top = tk.Frame(self.root)
        top.pack(fill=tk.X, padx=8, pady=6)
        tk.Label(top, text="Usuario:").pack(side=tk.LEFT)
        user_menu = tk.OptionMenu(top, self.current_user, *self.available_users)
        user_menu.pack(side=tk.LEFT)
        tk.Button(top, text="Crear usuario", command=self.create_user).pack(side=tk.LEFT, padx=6)
        mid = tk.Frame(self.root)
        mid.pack(fill=tk.BOTH, expand=True, padx=8, pady=6)
        left = tk.Frame(mid)
        left.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
        tk.Label(left, text="Archivos").pack()
        self.listbox = tk.Listbox(left)
        self.listbox.pack(fill=tk.BOTH, expand=True)
        self.listbox.bind('<Double-1>', lambda e: self.open_file())
        btns = tk.Frame(self.root)
        btns.pack(fill=tk.X, padx=8, pady=6)
        tk.Button(btns, text="Crear", command=self.create_file).pack(side=tk.LEFT)
        tk.Button(btns, text="Abrir", command=self.open_file).pack(side=tk.LEFT)
        tk.Button(btns, text="Modificar", command=self.modify_file).pack(side=tk.LEFT)
        tk.Button(btns, text="Eliminar", command=self.delete_file).pack(side=tk.LEFT)
        tk.Button(btns, text="Papelera", command=self.view_trash).pack(side=tk.LEFT)
        tk.Button(btns, text="Recuperar", command=self.recover_file).pack(side=tk.LEFT)
        tk.Button(btns, text="Asignar permisos",
command=self.assign_permissions).pack(side=tk.LEFT)
```

```
        tk.Button(btns, text="Refrescar", command=self.refresh_list).pack(side=tk.RIGHT)

    def create_user(self):
        name = simpledialog.askstring("Crear usuario", "Nombre del nuevo usuario:")
        if not name:
            return
        if name in self.available_users:
            messagebox.showinfo("Info", "Usuario ya existe")
            return
        self.available_users.append(name)
                                                                menu                    =
self.root.nametowidget(self.root.winfo_children()[0].winfo_children()[2].winfo_name())
        self.current_user.set(name)
        self.refresh_list()
    def refresh_list(self):
        self.listbox.delete(0, tk.END)
        files = fat.list_files()
        for e in files:
            label = f"{e.get('name')} (owner: {e.get('owner')})"
            self.listbox.insert(tk.END, label)
    def _get_selected_id(self, include_trash=False):
        idx = self.listbox.curselection()
        if not idx:
            messagebox.showwarning("Atención", "Seleccione un archivo")
            return None
```

```
        label = self.listbox.get(idx)
        name = label.split(" (owner:")[0]
        files = fat.list_files(include_trash=include_trash)
        for e in files:
            if e.get("name") == name:
                return e.get("id")
        return None
    def create_file(self):
        name = simpledialog.askstring("Crear archivo", "Nombre del archivo:")
        if not name:
            return
        content = simpledialog.askstring("Contenido", "Ingrese el contenido del archivo:")
        if content is None:
            return
        owner = self.current_user.get()
        fid = fat.create_file(name, content, owner)
        messagebox.showinfo("Creado", f"Archivo creado con id {fid}")
        self.refresh_list()
    def open_file(self):
        fid = self._get_selected_id()
        if not fid:
            return
        user = self.current_user.get()
        try:
```

```
            entry, content = fat.read_file(fid, user)
        except PermissionError:
            messagebox.showerror("Error", "No tiene permiso de lectura")
            return
        except FileNotFoundError:
            messagebox.showerror("Error", "Archivo no encontrado")
            return
        top = tk.Toplevel(self.root)
        top.title(entry.get("name"))
        meta = tk.Text(top, height=10)
        meta.insert(tk.END, json.dumps(entry, indent=2, ensure_ascii=False))
        meta.config(state=tk.DISABLED)
        meta.pack(fill=tk.BOTH)
        content_box = tk.Text(top)
        content_box.insert(tk.END, content)
        content_box.config(state=tk.DISABLED)
        content_box.pack(fill=tk.BOTH, expand=True)
    def modify_file(self):
        fid = self._get_selected_id()
        if not fid:
            return
        user = self.current_user.get()
        try:
            entry, content = fat.read_file(fid, user)
```

```
        except PermissionError:
            messagebox.showerror("Error", "No tiene permiso de lectura/escritura")
            return
        except FileNotFoundError:
            messagebox.showerror("Error", "Archivo no encontrado")
            return
        new = simpledialog.askstring("Modificar", "Nuevo contenido:", initialvalue=content)
        if new is None:
            return
        try:
            fat.modify_file(fid, new, user)
            messagebox.showinfo("OK", "Archivo modificado correctamente")
            self.refresh_list()
        except PermissionError:
            messagebox.showerror("Error", "No tiene permiso de escritura")
    def delete_file(self):
        fid = self._get_selected_id()
        if not fid:
            return
        user = self.current_user.get()
        try:
            fat.delete_file(fid, user)
            messagebox.showinfo("OK", "Archivo movido a papelera")
            self.refresh_list()
```

```
        except FileNotFoundError:
            messagebox.showerror("Error", "Archivo no encontrado")
    def view_trash(self):
        trash_win = tk.Toplevel(self.root)
        trash_win.title("Papelera")
        lb = tk.Listbox(trash_win)
        lb.pack(fill=tk.BOTH, expand=True)
        files = fat.list_files(include_trash=True)
        trash_items = [e for e in files if e.get('trash')]
        for e in trash_items:
            lb.insert(tk.END, f"{e.get('name')} (owner: {e.get('owner')})")
        def open_trash():
            sel = lb.curselection()
            if not sel:
                messagebox.showwarning("Atención", "Seleccione un archivo")
                return
            label = lb.get(sel)
            name = label.split(" (owner:")[0]
            for entry in trash_items:
                if entry.get('name') == name:
                    try:
                        ent, content = fat.read_file(entry.get('id'), self.current_user.get())
                    except PermissionError:
                        messagebox.showerror("Error", "No tiene permiso para leer este archivo")
```

```
                return
            top = tk.Toplevel(trash_win)
            top.title(entry.get('name'))
            t = tk.Text(top)
            t.insert(tk.END, content)
            t.config(state=tk.DISABLED)
            t.pack(fill=tk.BOTH, expand=True)
        tk.Button(trash_win, text="Abrir", command=open_trash).pack(side=tk.LEFT)
    def recover_file(self):
        fid = self._get_selected_id(include_trash=True)
        if not fid:
            return
        try:
            fat.recover_file(fid, self.current_user.get())
            messagebox.showinfo("OK", "Archivo recuperado")
            self.refresh_list()
        except FileNotFoundError:
            messagebox.showerror("Error", "Archivo no encontrado")
    def assign_permissions(self):
        fid = self._get_selected_id()
        if not fid:
            return
        owner = fat.get_entry(fid).get("owner")
        if self.current_user.get() != owner:
```

```
            messagebox.showerror("Error", "Solo el owner puede asignar permisos")
            return
        target = simpledialog.askstring("Permisos", "Usuario al que dar/quitar permisos:")
        if not target:
            return
        read = messagebox.askyesno("Lectura", "Permitir lectura?")
        write = messagebox.askyesno("Escritura", "Permitir escritura?")
        try:
            fat.set_permission(fid, owner, target, read, write)
            messagebox.showinfo("OK", "Permisos actualizados")
        except PermissionError:
            messagebox.showerror("Error", "No autorizado")
if __name__ == '__main__':
    root = tk.Tk()
    app = App(root)
    root.mainloop()
```

Introducción

El siguiente programa simula el funcionamiento básico de un sistema de archivos FAT (File Allocation Table), un método utilizado en sistemas operativos para administrar archivos en disco. El sistema permite crear, modificar, leer, eliminar y recuperar archivos, además de asignar permisos a diferentes usuarios.

La simulación se implementa con Python 3, utilizando los módulos estándar os, json, uuid, datetime y la biblioteca gráfica tkinter para la interfaz de usuario.

Estructura general del programa

El código se divide en dos grandes partes:

- **Gestor del sistema de archivos (FATManager)**: maneja la lógica del sistema FAT (almacenamiento, bloques, permisos, etc.).
- **Interfaz gráfica (App)**: permite interactuar visualmente con el sistema mediante botones y cuadros de diálogo.

Configuración inicial

```
DATA_DIR = "fat_data"

BLOCKS_DIR = os.path.join(DATA_DIR, "blocks")

FAT_PATH = os.path.join(DATA_DIR, "fat_table.json")
```

Estas líneas crean las carpetas necesarias para almacenar los datos:

- fat_data: carpeta principal del sistema.
- blocks: donde se guardan los bloques individuales de cada archivo.
- fat_table.json: archivo JSON que actúa como la tabla FAT, guardando la información de cada archivo.

Si las carpetas no existen, el código las crea automáticamente con os.makedirs().

Clase FATManager

Esta clase representa el **núcleo del sistema FAT**.
Cada archivo se identifica con un **UUID** (identificador único) y se almacena dividido en **bloques JSON** que contienen fragmentos de datos.

Principales métodos:

- **load() y save()**: cargan o guardan la tabla FAT desde/hacia el archivo JSON.
- **create_file(name, content, owner)**: crea un nuevo archivo. Divide el contenido en bloques de 20 caracteres y guarda cada bloque como un archivo JSON dentro de blocks/.
- **_create_blocks()**: función interna que fragmenta el contenido y crea los bloques enlazados entre sí (como una lista enlazada simple).
- **read_file(file_id, user)**: reconstruye y devuelve el contenido del archivo leyendo cada bloque consecutivo.
- **modify_file(file_id, new_content, user)**: reemplaza el contenido del archivo, borra los bloques antiguos y genera nuevos.
- **delete_file(file_id, user)**: marca un archivo como eliminado (lo mueve a la "papelera").
- **recover_file(file_id, user)**: restaura un archivo eliminado.
- **set_permission(file_id, owner, target, read, write)**: permite al dueño del archivo asignar permisos de lectura/escritura a otros usuarios.
- **list_files(include_trash=False)**: lista los archivos existentes, pudiendo incluir o no los eliminados.

Cada entrada de archivo en la FAT contiene:

```
{
 "id": "UUID",
 "name": "nombre.txt",
 "data_path": "ruta del primer bloque",
 "trash": false,
 "size": 150,
 "created": "fecha creación",
 "modified": "fecha modificación",
 "deleted": null,
 "owner": "usuario",
```

```
 "permissions": {
   "usuario": {"read": true, "write": true}
 }
}
```

Clase App (Interfaz Gráfica)

La clase App crea una ventana con Tkinter donde los usuarios pueden realizar las operaciones del sistema FAT sin usar la consola.

Componentes principales:

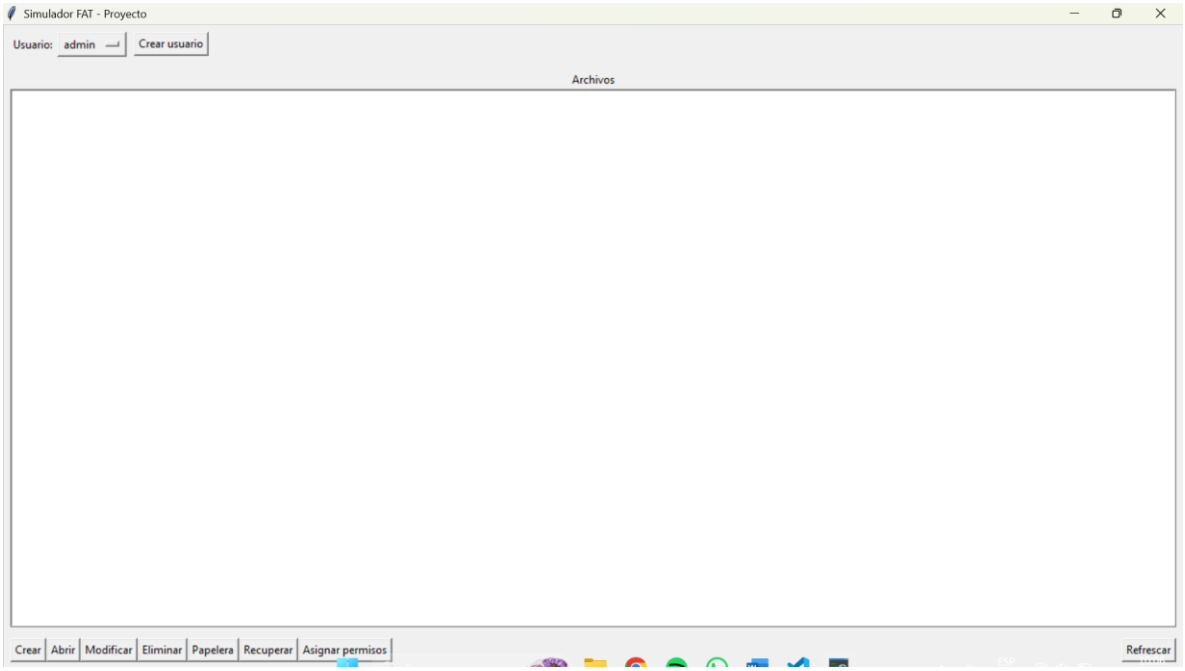
- **Menú superior**: muestra el usuario actual y permite crear nuevos usuarios.
- **Lista de archivos**: muestra los archivos disponibles y sus dueños.
- **Botones de control**:
- Crear, Abrir, Modificar, Eliminar, Papelera, Recuperar, Asignar permisos, Refrescar lista.

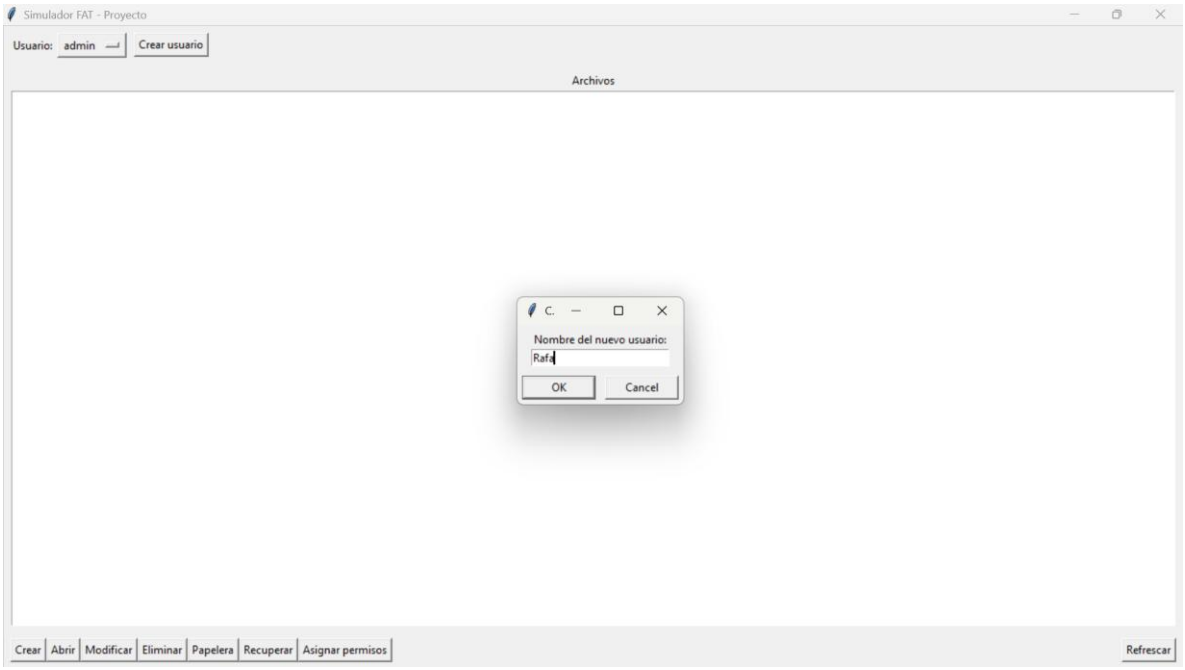
Flujo de interacción:

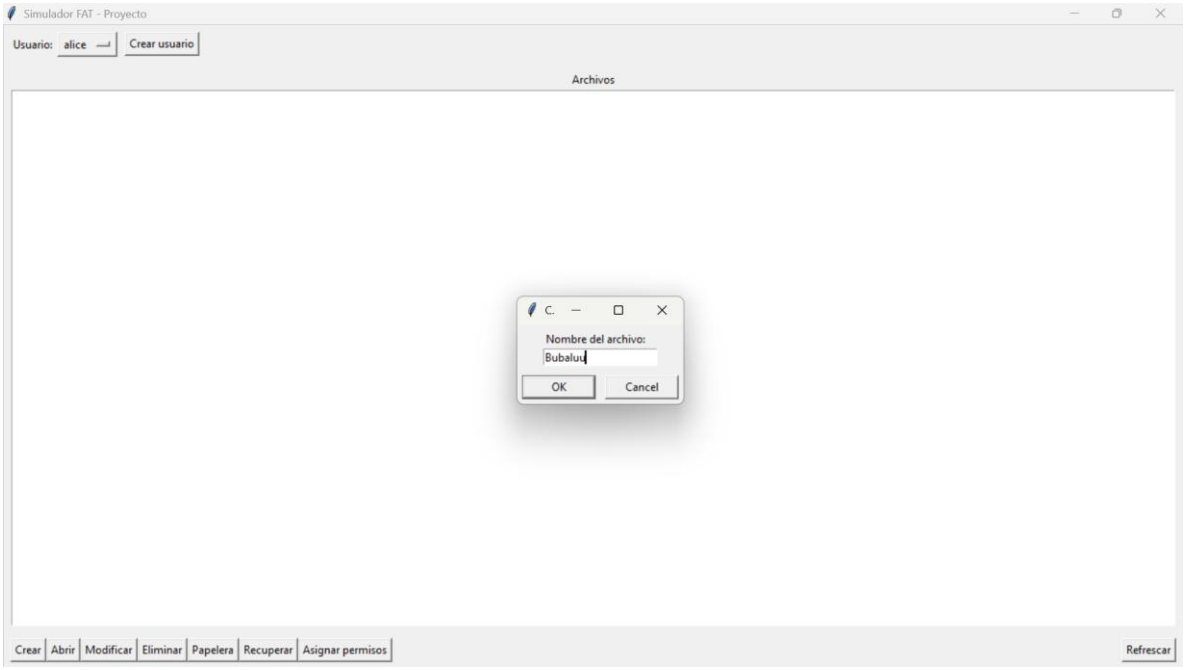
- El usuario selecciona su nombre o crea uno nuevo.
- Puede crear un archivo nuevo ingresando nombre y contenido.
- Puede abrir archivos (si tiene permisos), modificarlos o eliminarlos.
- Los archivos eliminados pasan a la "papelera", desde donde pueden abrirse o recuperarse.
- El propietario de un archivo puede conceder o revocar permisos de lectura/escritura a otros usuarios.

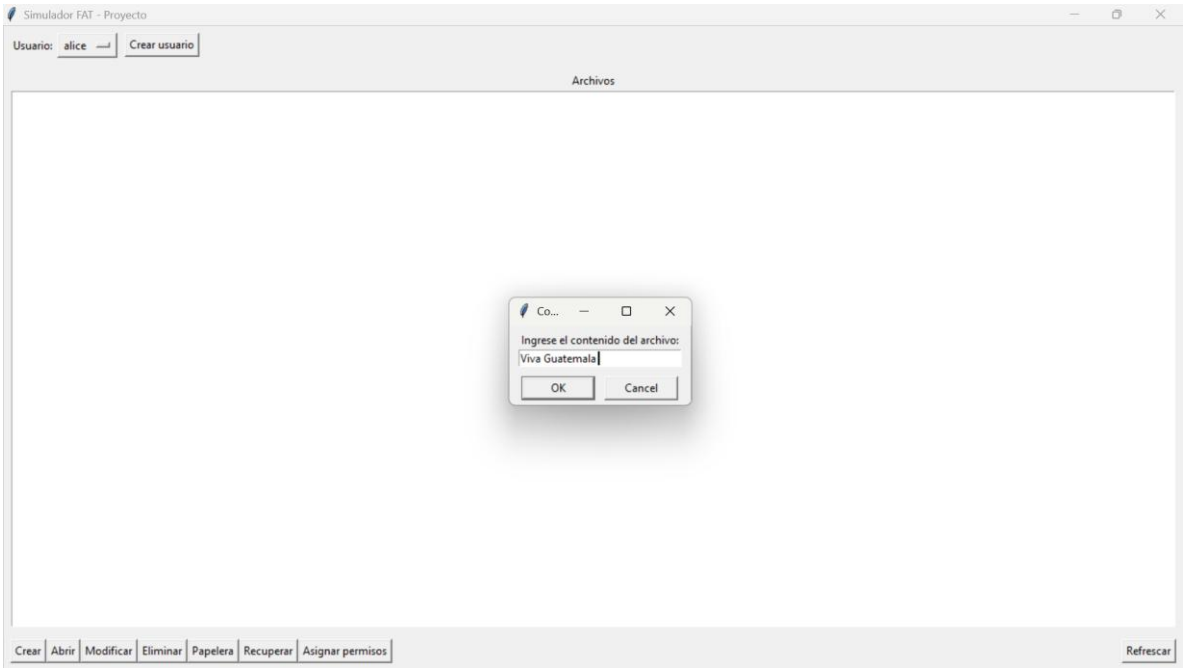
Funcionamiento general

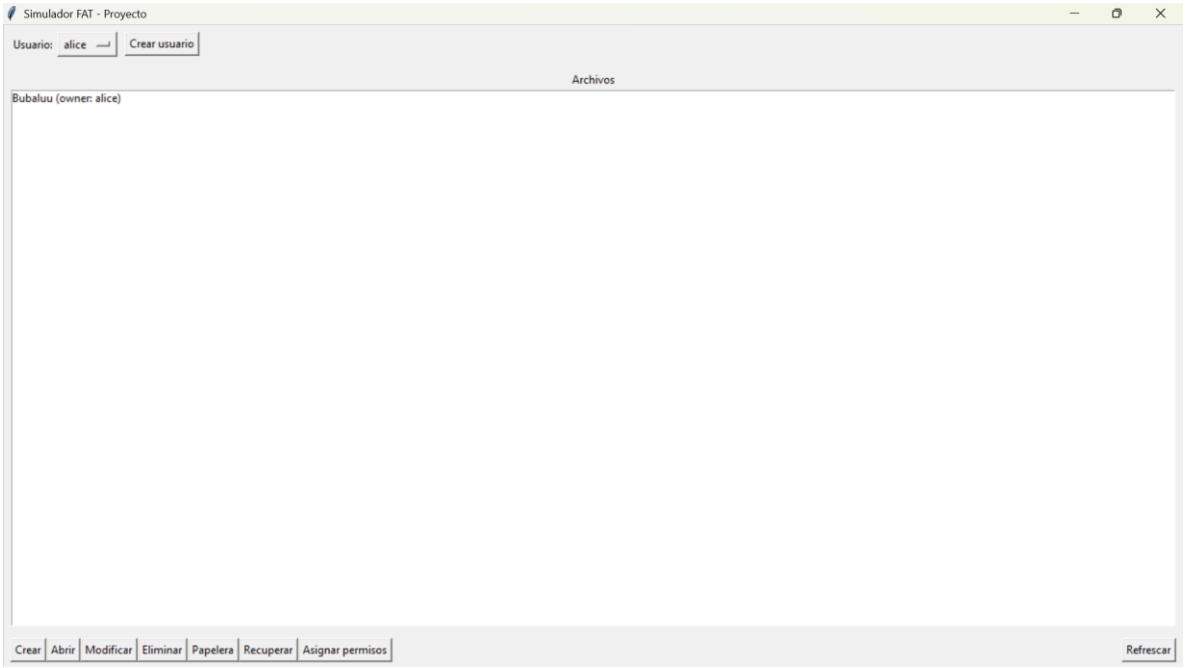
- Al crear un archivo, su contenido se divide en partes de 20 caracteres.
- Cada parte se guarda como un bloque JSON enlazado al siguiente (emulando cómo FAT enlaza sectores en disco).
- La tabla fat_table.json guarda los metadatos y permisos de cada archivo.
- Los usuarios interactúan mediante la interfaz gráfica, que llama internamente a los métodos de FATManager.

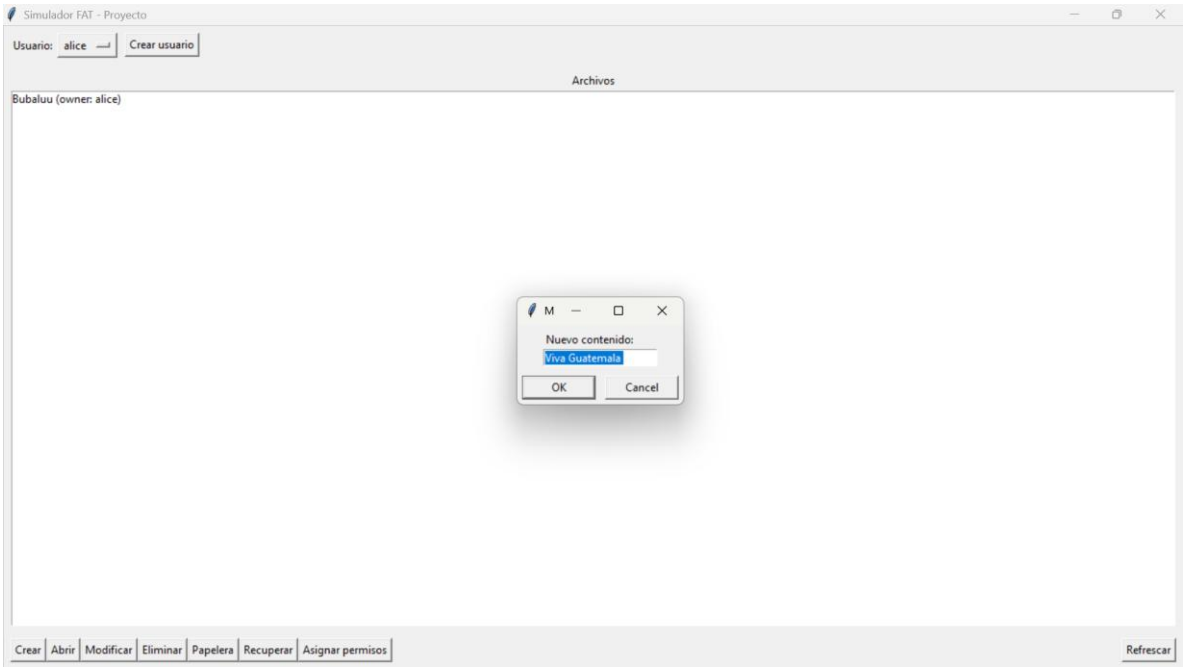

Simulador FAT - Proyecto
Usuario:
admin
Crear usuario
Archivos
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar

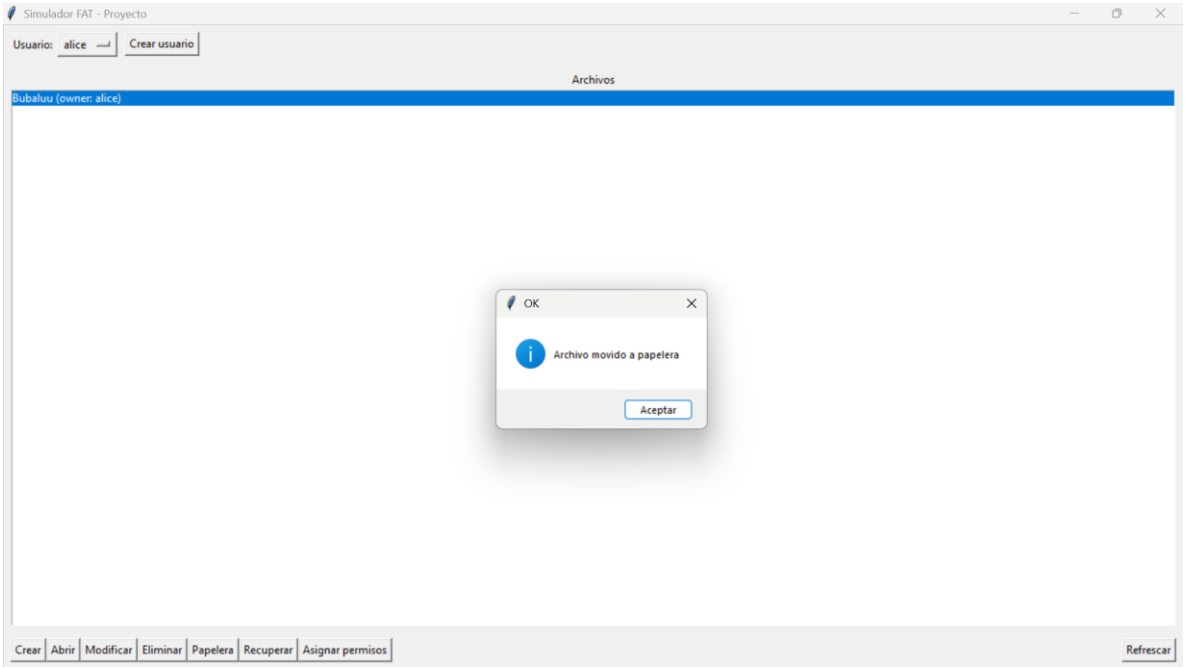

Simulador FAT - Proyecto
Usuario:
admin
Crear usuario
Archivos
C...
Nombre del nuevo usuario:
Rafa
OK
Cancel
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar

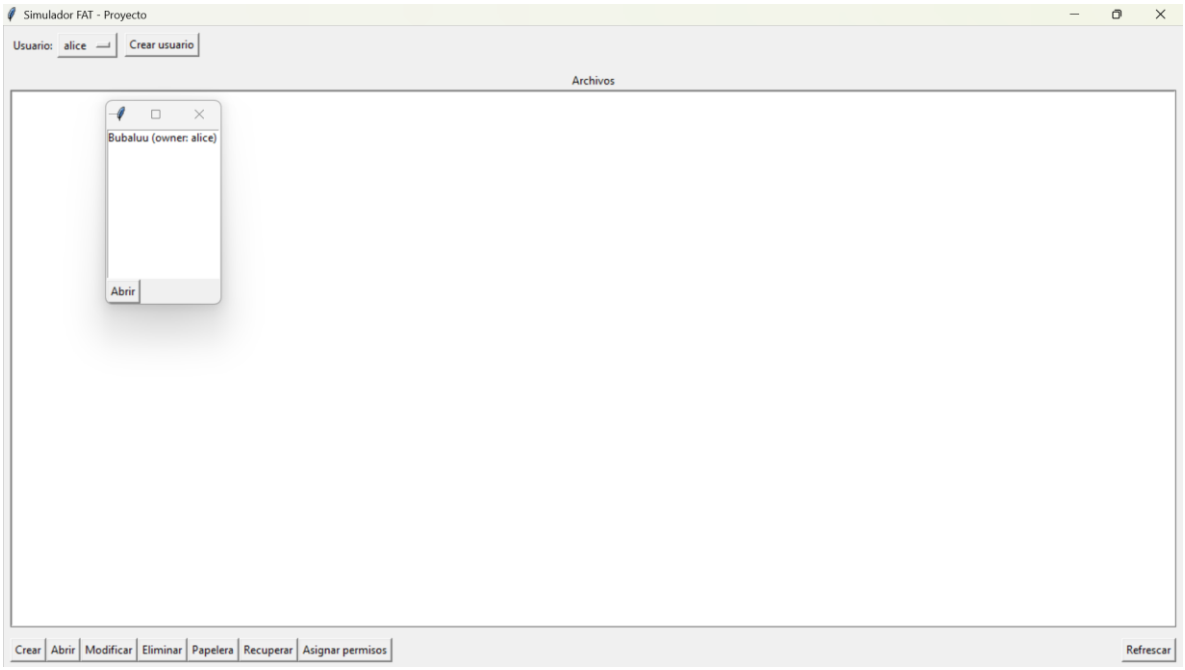

Simulador FAT - Proyecto
Usuario: alice
Crear usuario
Archivos
C...
Nombre del archivo:
Bubaluu
OK
Cancel
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar

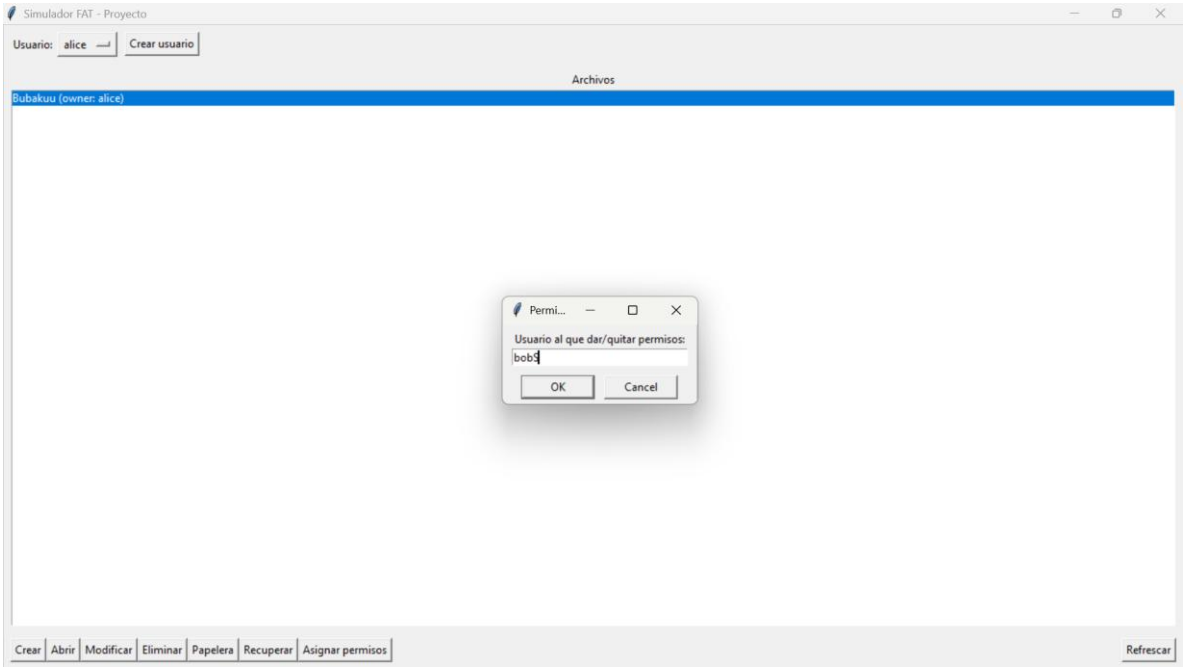

Simulador FAT - Proyecto
Usuario: alice
Crear usuario
Archivos
Co...
Ingrese el contenido del archivo:
Viva Guatemala
OK
Cancel
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar

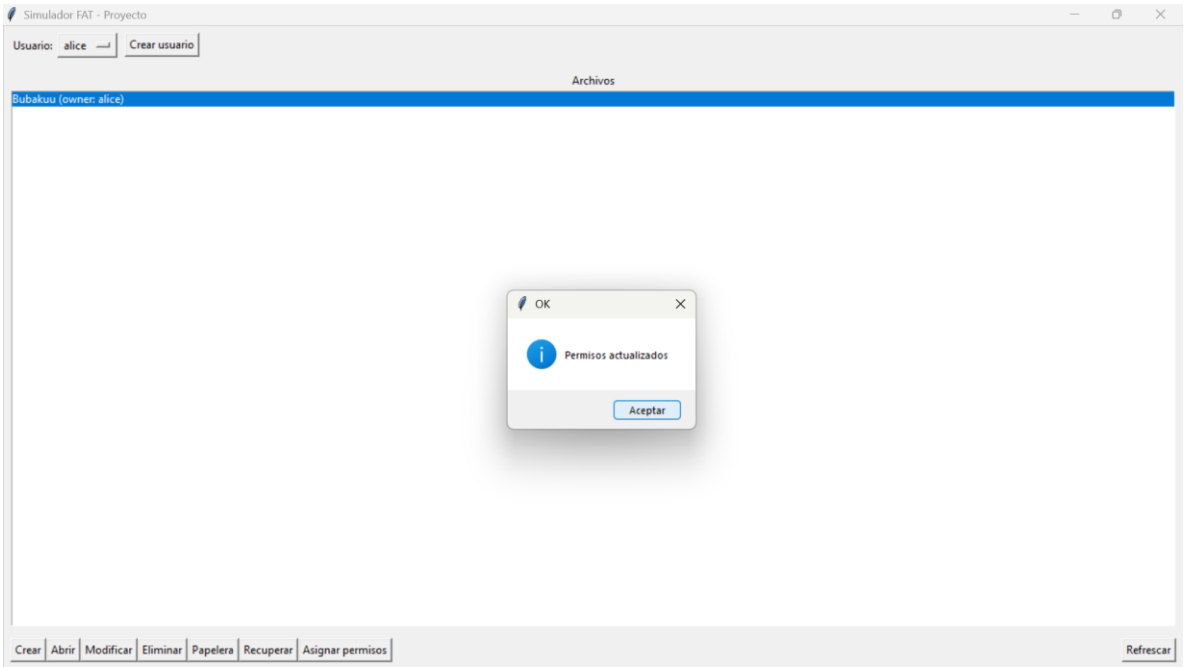

Simulador FAT - Proyecto
Usuario:
alice
Crear usuario
Archivos
Bubaluu (owner: alice)
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar


Simulador FAT - Proyecto
Usuario:
alice
Crear usuario
Archivos
Bubaluu (owner: alice)
M
Nuevo contenido:
Viva Guatemala
OK
Cancel
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar


Simulador FAT - Proyecto
Usuario:
alice
Crear usuario
Archivos
Bubaluu (owner: alice)
OK
Archivo movido a papelera
Aceptar
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar


Simulador FAT - Proyecto
Usuario:
alice
Crear usuario
Archivos
Bubaluu (owner: alice)
Abrir
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar


Simulador FAT - Proyecto
Usuario: alice
Crear usuario
Archivos
Bubakuu (owner: alice)
Permi...
Usuario al que dar/quitar permisos:
bob$
OK
Cancel
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar


Simulador FAT - Proyecto
Usuario: alice
Crear usuario
Archivos
Bubakuu (owner: alice)
OK
Permisos actualizados
Aceptar
Crear
Abrir
Modificar
Eliminar
Papelera
Recuperar
Asignar permisos
Refrescar