Remote File System in Rust – Specification

1. Overview

This project aims to implement a **remote file system client** in Rust that presents a local mount point, mirroring the structure and contents of a file system hosted on a remote server. The file system should support **transparent read and write access** to remote files and directories.

2. Goals

- Provide a local file system interface that interacts with a remote storage backend.
- Enable standard file operations (read, write, create, delete, etc.) on remote files as if they were local.
- Ensure compatibility with Linux systems.
- Optionally support Windows and macOS with best-effort implementation.

3. Functional Requirements

3.1 Core Functionality

☒ Mount a virtual file system to a local path (e.g., `/mnt/remote-fs`)

☒ Display directories and files from a remote source

☒ Read files from the remote server

☒ Write modified files back to the remote server

☒ Support creation, deletion, and renaming of files and directories

☒ Maintain file attributes such as size, timestamps, and permissions (as feasible)

☒ Run as a background **daemon** process that handles filesystem operations continuously

3.2 Server Interface and implementation

- The server should offer a set RESTful API for file operations:
  - `GET /list/<path>` – List directory contents
  - `GET /files/<path>` – Read file contents
  - `PUT /files/<path>` – Write file contents
  - `POST /mkdir/<path>` – Create directory
  - `DELETE /files/<path>` – Delete file or directory
- The server can be implemented using any language or framework, but should be RESTful and stateless.

3.3 Caching

- Optional local caching layer for performance
- Configurable cache invalidation strategy (e.g., TTL or LRU)

4. Non-Functional Requirements

4.1 Platform Support

- ✅ **Linux** – Full support using FUSE (`libfuse`, `fuser`, or `async-fuse`)
- ⚪ **macOS** – Optional support using macFUSE (best effort, no guarantee of full stability)
- ⚪ **Windows** – Optional support using WinFSP or Dokany with C bindings (lower priority)

4.2 Performance

- Support for large files (100MB+) with streaming read/write
- Reasonable latency (<500ms for operations under normal network conditions)

4.3 Startup and Shutdown

- Graceful startup and shutdown procedures