CloudBox: A Full-Stack Web-Based Personal File Manager

Nilesh Chakrabarty
Enrollment: 230053
Rishihood University, India
nilesh.c23csai@nst.rishihood.edu.in

Nikhil Raj
Enrollment: 230080
Rishihood University, India
nikhil.r23csai@nst.rishihood.edu.in

***Abstract*—This paper presents the design and implementation of CloudBox, a full-stack, web-based personal file manager featuring a modern dark user interface. The system supports a comprehensive suite of file management operations, including uploading, organizing, previewing, starring, and searching files, all backed by a real filesystem. Key features include a soft-delete mechanism via a Trash directory, live storage statistics, and type-aware file previews for media and text. The frontend is built using Next.js, React, and Tailwind CSS, providing a responsive and dynamic user experience without page reloads. The backend leverages Node.js and Express to handle asynchronous filesystem operations and multipart file uploads. This document details the system architecture, feature set, application programming interface (API) specifications, and the fundamental design decisions that enable efficient local file organization and retrieval.**

***Index Terms*—File Manager, Full-Stack, Web Application, Cloud Storage, Next.js**

I. Introduction

CloudBox is a comprehensive, full-stack personal file management system designed to operate via a web browser. It provides users with an intuitive, dark-themed interface to manage files on a local or remote filesystem. The system bridges the gap between traditional desktop file explorers and modern web interfaces by offering instant navigation, inline editing, and robust file preview capabilities.

II. System Features

The application supports a wide array of features designed for seamless file manipulation and organization.

A. Core Management

Users can browse directories using breadcrumb navigation and toggle between list and grid views. The system allows drag-and-drop file uploads with progress feedback, instant directory creation, and inline renaming where the input automatically selects the current name. Files can be moved across the directory tree using a dedicated path picker.

B. File Organization and Search

Important files can be bookmarked and accessed via a dedicated "Starred" view. The system includes a global, full-name search across all files and folders, deliberately excluding hidden system files. A dynamic "Recent Files" view automatically tracks previously accessed items. Furthermore, a categorized view groups files by their inherent types (e.g., documents, images, audio, video, code).

C. Data Safety and Previews

A soft-delete mechanism sends removed items to a Trash folder, allowing for one-click restoration to their original paths, or permanent deletion. The UI also features a type-aware preview dialog capable of rendering images, video, audio, PDFs via iframes, rendered Markdown, and plain-text files with syntax highlighting and line numbers. A live storage widget tracks used versus total storage against a configurable quota.

III. Technology Stack

The architecture is divided into a decoupled frontend and backend, ensuring modularity and performance.

A. Frontend Architecture

The client side is powered by Next.js (16.1.6) utilizing the App Router, with React (19.2.3) serving as the core UI library. TypeScript ensures robust type safety across components. Styling is handled via Tailwind CSS, while Radix UI provides accessible primitive components (e.g., dialogs). Framer Motion handles fluid layout animations, and Lucide React supplies the iconography.

B. Backend Architecture

The server operates on Node.js ($\geq$ 20) utilizing Express for HTTP routing. File upload handling is managed by the Multer middleware. Core filesystem interactions rely on the native `fs/promises` module to ensure non-blocking, asynchronous operations.

IV. Project Structure

The repository is split into frontend and backend directories to maintain separation of concerns.

The backend contains `server.js`, which exposes 20 RESTful endpoints, and a `demo-files` directory acting as the root storage. This directory contains standard folders (Audio, Documents, etc.) alongside hidden application state folders like `.trash/` and `.starred.json`.

The frontend encapsulates Next.js routing within the `app/page.tsx` root, which handles all state and view routing. The `components/` directory holds modular UI elements such as the sidebar, context bar, file explorer, and various dialogs. API interactions are abstracted within `lib/api.ts` using strongly typed fetch wrappers.

V. API Reference

The backend exposes a comprehensive REST API operating on port 5002.

TABLE I
FILESYSTEM ENDPOINTS

Endpoint	Description
`GET /api/files`	List directory contents
`GET /api/files/all`	Recursive list of all files
`POST /api/files/upload`	Upload file (multipart/form-data)
`POST /api/folders/create`	Create a new folder
`PATCH /api/files/rename`	Rename a file or folder
`POST /api/files/move`	Move a file or folder
`DELETE /api/files/delete`	Soft-delete or permanently delete

TABLE II
SYSTEM & METADATA ENDPOINTS

Endpoint	Description
`GET /api/files/trash`	List Trash contents
`POST /api/files/restore`	Restore an item from Trash
`GET /api/files/starred`	List starred files
`GET /api/storage`	Retrieve storage usage stats

VI. Key Design Decisions

Several architectural choices were made to optimize user experience and performance:

- *Soft Delete Mechanism*: Files moved to Trash are stored in `.trash/` with timestamped prefixes and a JSON metadata file to retain the original path for accurate restoration.
- *Hidden State Management*: Application state files (`.trash/`, `.starred.json`) are filtered globally from API responses, preventing them from rendering in the UI.
- *Single-Page Application (SPA) Routing*: All views (My Files, Recent, Starred, Trash, Search) are controlled via a single `viewMode` React state. This eliminates HTTP routing delays, making navigation instantaneous.
- *Storage Cap*: A configurable 2 GB storage limit is enforced on the backend and visually represented via a live progress widget on the frontend.

VII. Conclusion

CloudBox demonstrates a robust approach to building a browser-based filesystem interface. By leveraging modern web technologies like Next.js and Node.js, it successfully replicates the efficiency of native file explorers while adding cloud-centric features like tagging, live previews, and soft deletion.

References

[1] Next.js Documentation, Vercel Inc. [Online]. Available: https://nextjs.org/docs

[2] Node.js File System API, OpenJS Foundation. [Online]. Available: https://nodejs.org/api/fs.html