FIRST EVALUATION PHASE

File Organizer System

A Trust-Centered macOS File Management Application

Team Lemon

Kalash · Soniya · Aryan

Platform	Language	Framework
macOS 13+	Swift 5.9	SwiftUI

PRESENTATION OVERVIEW

What We'll Cover

01

Problem Relevance

Why this is an OS problem

02

Objectives

Goals and philosophy

03

Feasibility

Scope and achievability

04

Implementation

What has been built

05

OS Concepts

Principles applied

06

Architecture

System design

07

Code Structure

Modularity

08

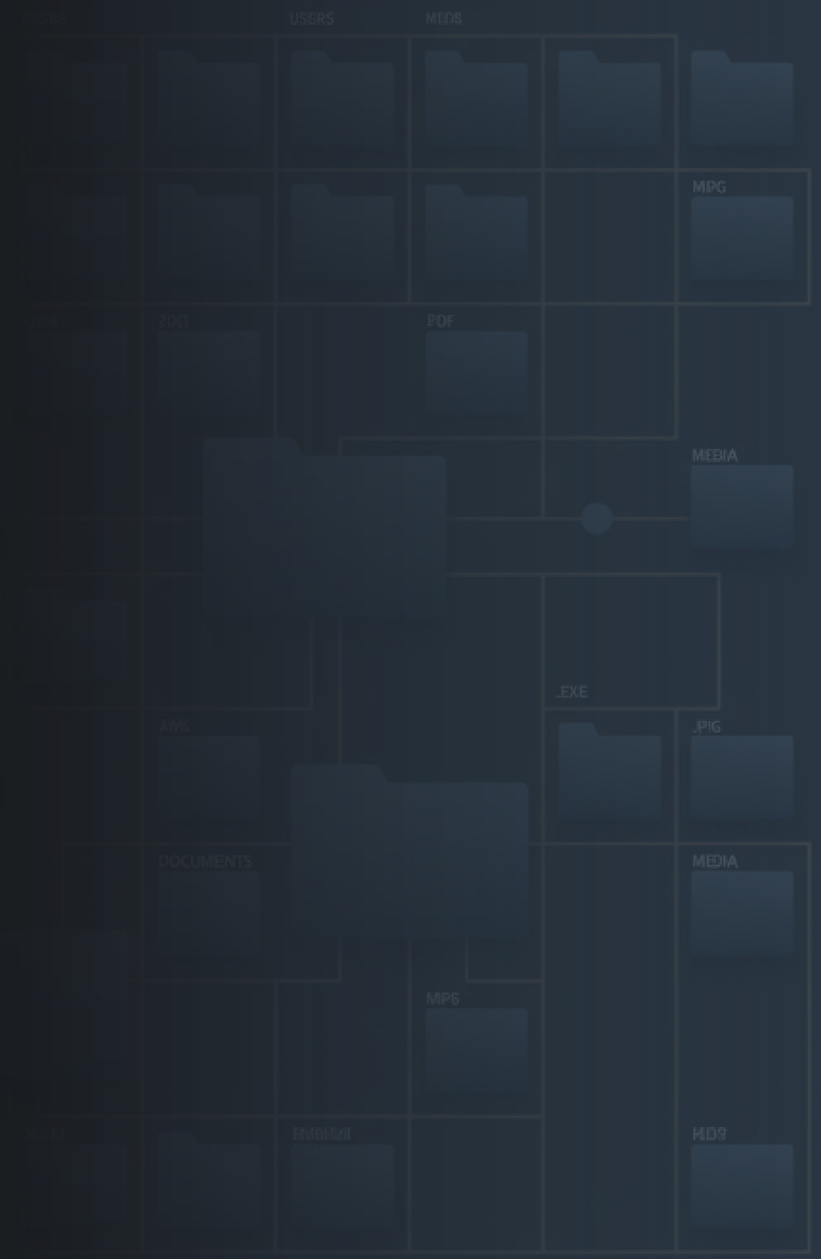
Achievements

Key highlights

CHAPTER 01

Problem Relevance

Why file organization is fundamentally an Operating Systems problem



PROBLEM RELEVANCE

Operating Systems Context

Filesystem Hierarchies

Recursive directory enumeration, path resolution, symlink handling

Metadata Management

Inode attributes, timestamps, permissions via OS APIs

I/O Operations

Move, copy, delete with atomicity and error handling

Permission Systems

Sandboxing, security-scoped resources, user consent

Key Insight: File organization is about safely orchestrating filesystem operations in a hostile, concurrent environment

CHAPTER 02

Project Objectives

Clear goals and design philosophy

CORE SYSTEM GOAL

What the System Exists to Do

Make filesystem changes that are

explainable, **reversible**, and **unsurprising**

Explainable

Every action has a clear reason

Reversible

All actions can be undone

Unsurprising

Nothing happens without approval

> Correctness and trust take precedence over convenience

CHAPTER 03

Feasibility & Scope

What's being built and why it's achievable

PROJECT SCOPE

What We're Building

User-Space Operations

All operations run in user space using public macOS APIs

Rule-Based Organization

Users define rules (conditions → outcomes)

Five Core Subsystems

Scanner, Planner, Executor, Undo, Observer

FSEvents Monitoring

Real-time filesystem change detection

Trust-Centered UI

Explicit Approval

Transparency

Reversible

Conservative

CHAPTER 04

Feasibility & Scope

What's being built and why it's achievable within the constraints of the course

PROJECT SCOPE

What We're Building

System Boundaries

- **User-Space Filesystem Operations**
  All operations run in user space using public macOS APIs. No kernel extensions or privileged operations required.
- **Declarative Rule-Based Organization**
  Users define rules (conditions → outcomes) that the system evaluates against files. Rules express intent, not procedures.
- **Five Core Subsystems**
  Scanner, Planner, Executor, Undo Engine, and Observer—each with strictly defined responsibilities.
- **FSEvents-Based Monitoring**
  Real-time filesystem event monitoring using macOS's native FSEvents API for detecting changes.

What's Out of Scope

- ✕ **Network Filesystems**
  NFS, SMB, and remote storage are not supported
- ✕ **Cloud Storage**
  iCloud Drive, Dropbox, Google Drive integration
- ✕ **Content Analysis**
  No file content inspection—only metadata-based rules
- ✕ **Automatic Execution**
  No background automation or scheduled tasks

Trust-Centered UI Philosophy

Explicit Approval
Nothing happens without user review and approval

Complete Transparency
Every action explained before and after execution

Reversible Actions
All destructive operations can be undone safely

Conservative Failures
Skip and explain rather than guess and risk

Platform Requirements

Operating System	macOS 13.0+
Swift Version	Swift 5.9+
Framework	SwiftUI
Concurrency	Swift Actors

CHAPTER 05

OS Concepts Applied

Correct application of Operating Systems principles

OS CONCEPTS

Filesystem & I/O Management

Asynchronous I/O

Task.detached for non-blocking scans. Actor-based serialization.

Security-Scoped Resources

Sandbox-compliant file access with bookmark persistence.

Metadata Caching

URLResourceKey for efficient batch metadata retrieval.

Error Handling

Graceful handling of permission denied, file not found, collisions.

OS CONCEPTS

Concurrency & Thread Safety

Swift Actors

Automatic serialization of mutable state access. No manual locks needed.

Sendable Protocol

Compile-time verification of thread-safe data transfer.

MainActor

UI state updates confined to main thread for safety.

Immutable Data

Structs preferred over classes. Value semantics for safety.

Thread Safety Guarantees

- No Data Races
- Compile-Time Check
- Main Thread UI
- Immutable Preferred

OS CONCEPTS

File System Monitoring

FSEvents API Integration

Kernel-level filesystem monitoring with proper C-to-Swift callback bridging.

Event Types

- .created
- .removed
- .modified
- .renamed

Limitations

- Events may be coalesced
- Events arrive with latency
- Events may be duplicated
- File may not exist when processed

Philosophy: Events are hints, not facts. User decides. No auto-execution.

OS CONCEPTS

Transaction & Recovery

ActionPlan

Immutable intent transaction

ExecutionLog

Durable audit record

UndoEngine

Best-effort recovery

Recovery Flow

1. Original action: A → B
2. Log entry created
3. Undo attempt: B → A
4. UndoLog entry

ACID Properties

Atomicity	✓
Consistency	✓
Isolation	✓
Durability	✓

Idempotent operations. Re-execution doesn't repeat effects.

CHAPTER 06

Implementation Progress

What has been built so far and the current state of each subsystem

Thank-you