Secure File Upload System

1.Design Explanation

The Secure File Upload System is designed to securely handle user file uploads while enforcing centralized security and compliance policies using policy-as-code principles.

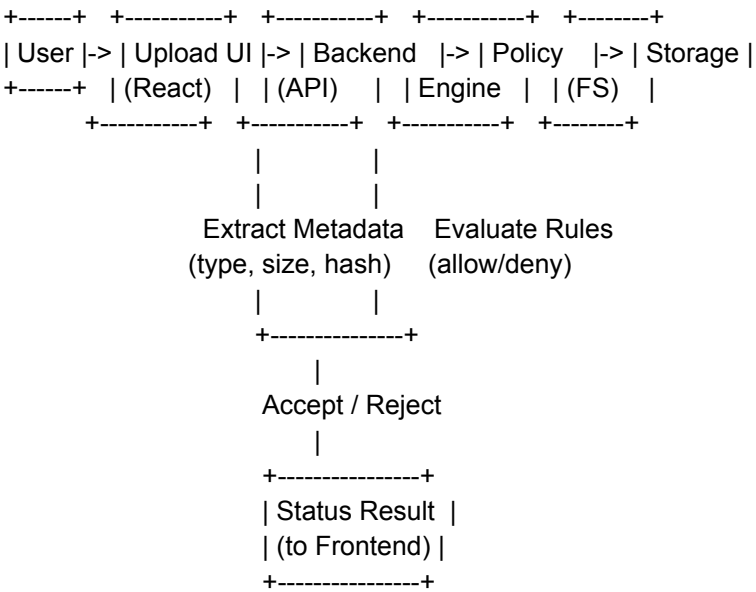
The system follows a backend-driven security model, where all validation and enforcement logic is executed server-side. The frontend is intentionally kept lightweight and untrusted, serving only as a user interface.

High-Level Workflow

1. User uploads a file via the React frontend
2. Backend receives the file and extracts metadata (filename, size, hash)
3. Metadata is sent to Open Policy Agent (OPA) for evaluation
4. OPA returns an allow/deny decision
5. Backend stores the file in the appropriate directory
6. File status is returned to the frontend for admin visibility

2. Architecture & Data-Flow Diagram

Simple Architecture Diagram



2.Data Flow Explanation

1. Upload Request
   The frontend sends a multipart file upload request to the backend.

2. Metadata Processing
   The backend extracts:

   - Filename
   - File size
   - Cryptographic hash

3. Policy Evaluation
   Metadata is sent to OPA over HTTP for policy evaluation.

4. Decision Handling

   - If allowed → file stored in `accepted/`
   - If denied → file stored in `rejected/`

5. Response
   The backend returns the file status to the frontend dashboard.

3. Design Trade-offs

Technology / Decision	Why It Was Chosen	Advantages	Trade-offs / Limitations
FastAPI (Backend)	Modern Python framework with async support	High performance, built-in validation, automatic API docs	Slight learning curve compared to Flask
React (Frontend)	Dynamic UI and component-based design	Responsive UI, reusable components, clean state handling	More setup than plain HTML/JS

Open Policy Agent (OPA)	Centralized policy-as-code engine	Decoupled security logic, policies change without redeploy	Extra service dependency
Rego (Policy Language)	Declarative security rules	Expressive and readable policies, industry standard	Requires learning new syntax
Backend Policy Enforcement	Zero-trust frontend model	Prevents client-side bypass, stronger security	Increased backend responsibility
Local File Storage	Simple and fast for demo	Easy setup, no external dependencies	Not scalable or distributed
Metadata Extraction	Policy-based decision making	Enables fine-grained validation	Slight processing overhead
In-Memory Metadata Store	Simplicity for demonstration	Fast access, minimal setup	Data lost on restart
HTTP Communication with OPA	Loose coupling	Language-agnostic, flexible	Network latency
No Authentication	Focus on core functionality	Faster development, clarity	Not production-ready