Web Application: File Server Management (PHP + Go) - Documentação Técnica

Visão Geral

Objetivo: Permitir que usuários autorizados criem pastas e façam upload de arquivos diretamente em servidores corporativos, seguindo padrões de nomenclatura e regras de segurança, de forma centralizada e auditável.

Arquitetura (High-Level)

Camadas principais:
- Frontend: React / Next.js
- API / Backend: PHP (Laravel) - autenticação, RBAC, regras de negócio, auditoria
- File Engine: Go - execução de operações no filesystem remoto (SMB/SFTP/NFS)
- File Server: storage híbrido (local / cloud)
- Banco: PostgreSQL (usuários, permissões, audit log)
- Filas: Redis / Kafka (tarefas assíncronas)
- Antivirus: ClamAV (scan de uploads)

Diagrama (Mermaid / Fluxo)

Mermaid syntax (incluído para render no repositório/design tool):

```
flowchart LR A["Cliente (Frontend)"] --> B["API Gateway / Backend"]; B --> C["Validação de Autenticação"]; C --> D{"Rota da API?"}; D -->|"Criar Recurso"| E["Controller: Create"]; D -->|"Atualizar Recurso"| F["Controller: Update"]; D -->|"Consultar Dados"| G["Controller: Read"]; D -->|"Excluir Recurso"| H["Controller: Delete"]; E --> I["Service Layer"]; F --> I; G --> I; H --> I; I --> J["Repository / ORM"]; J --> K["Banco de Dados"]; K --> J; J --> I; I --> B; B --> A;
```

Modelo de Banco de Dados (SQL)

```
-- Usuários CREATE TABLE users ( id SERIAL PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, password_hash VARCHAR(255) NOT NULL, email VARCHAR(100) UNIQUE, role VARCHAR(50), created_at TIMESTAMP DEFAULT NOW(), updated_at TIMESTAMP DEFAULT NOW() ); -- Permissões de pastas CREATE TABLE permissions ( id SERIAL PRIMARY KEY, user_id INTEGER REFERENCES users(id) ON DELETE CASCADE, folder_path VARCHAR(255) NOT NULL, can_create BOOLEAN DEFAULT FALSE, can_upload BOOLEAN DEFAULT FALSE, can_delete BOOLEAN DEFAULT FALSE ); -- Auditoria CREATE TABLE audit_log ( id SERIAL PRIMARY KEY, user_id INTEGER REFERENCES users(id), action VARCHAR(50), -- 'create_folder', 'upload_file', etc folder_path VARCHAR(255), filename VARCHAR(255), timestamp TIMESTAMP DEFAULT NOW() );
```

Estrutura de Pastas do Projeto (sugestão)

```
project-root/ ■■ frontend/ # React / Next.js ■ ■■ components/ ■ ■■ pages/ ■ ■■ services/ ■■ backend/ # Laravel API ■ ■■ app/ ■ ■ ■■ Http/Controllers/ ■ ■ ■■ Services/ # Chama Go File Engine ■ ■ ■■ Policies/ ■ ■■ database/migrations/ ■ ■■ routes/ ■ ■■ tests/ ■■ file-engine-go/ # Go service ■ ■■ cmd/ # Entrypoint ■ ■■ internal/ ■ ■ ■■ filesystem/ ■ ■ ■■ validators/ ■ ■ ■■ uploader/ ■ ■■ pkg/ ■■ docker/ # Dockerfiles + Compose / Helm charts ■■ docs/
```

Endpoints de API - Exemplos (Requests / Responses)

1) Criar pasta (assíncrono via fila)

Request:

```
POST /api/folders Headers: Authorization: Bearer <token> Body (JSON): { "parent_path": "/projects/cliente-a/", "name": "cliente-a-2025-12", "metadata": { "owner": "team-x" } }
```

Response:

```
202 Accepted Body: { "status": "queued", "task_id": "uuid-1234", "message": "Pasta em
criação. Verifique o status via /tasks/{task_id}" }
```

2) Upload de arquivo (fluxo seguro)

Request / flow:

```
POST /api/uploads/initiate Headers: Authorization: Bearer <token> Body (JSON): {
"target_path": "/projects/cliente-a/cliente-a-2025-12/", "filename": "documento.pdf",
"size": 1234567 } Response: { "upload_url": "https://s3.temp/upload-presigned-url",
"upload_id": "uuid-upload-1" } -- Client uploads directly to S3 using upload_url -- After
upload, client calls /api/uploads/complete with upload_id
```

Complete Response:

```
200 OK { "status": "queued_scan", "message": "Arquivo recebido e enviado para scan. Após
aprovação, será movido para o File Server." }
```

Tabela de Regras de Naming (exemplo)

	Regex / Regra	Exemplo	Observações
	^[a-z0-9\-]+$	cliente-a	lowercase, hífen permi
	^[0-9]{4}$	2025	ano com 4 dígitos
	^(0[1-9]|1[0-2])$	12	MM
	cliente-ano-mes	cliente-a-2025-12	concatenar campos co

Tabela de Permissões (modelo)

Perfil	Pode Criar Pasta	Pode Upload	Pode Deletar	Observações
Admin	✔■	✔■	✔■	Acesso total
PowerUser	✔■	✔■	■	Pode criar e enviar, não pode deletar
Viewer	■	■	■	Somente leitura

Sequência: Criação de Pasta (texto UML/Sequence)

Usuário -> Frontend: solicita criação de pasta
Frontend -> API Laravel: POST /api/folders
API Laravel -> Validator: verifica regras de nome e permissão
API Laravel -> Queue (Redis/Kafka): Envia tarefa 'create_folder'
File Engine (Go) -> File Server: Cria pasta no caminho solicitado
File Engine (Go) -> API Laravel: Retorna status (success/fail)
API Laravel -> Audit Log DB: Registra entrada
API Laravel -> Frontend: Notifica usuário (via websocket ou polling)

Security Checklist (resumo)

Item	Descrição
Path traversal	Sanitizar e validar todos os paths. Nenhum input direto para filesystem.
Extensões permitidas	Whitelist: .pdf, .docx, .xlsx, .txt (configurável)
Scan de malware	ClamAV ou serviço similar para todos os uploads antes do move final.
RBAC	Policies no Laravel + acl no File Server quando aplicável.

Roadmap resumido

Fase 1 (MVP - 3 meses): Autenticação, listagem de pastas, criar pasta
Fase 2 (3-6 meses): Upload, fila, auditoria básica
Fase 3 (6-12 meses): ClamAV, regras de nome, permissões granulares
Fase 4 (12+ meses): Versionamento, analytics, alta disponibilidade

Observações finais

- Incluí o mermaid flowchart que foi gerado na ferramenta de diagrama. Se desejar, posso exportar o diagrama como PNG/SVG e incluir no PDF (requer export gráfico).
- Este PDF contém exemplos práticos que a equipe pode usar no desenvolvimento (endpoints, SQL, tabelas de regras).
- Se quiser que eu gere também arquivos adicionais (Postman collection, OpenAPI spec, ou diagramas PNG/SVG), eu posso criar esses artefatos também.