Frontend Research

> Scope: Ideation/research for the customer-facing and admin dashboards that ingest 360° sewer videos, track processing, and present resulting 3D models & reports. This document proposes the frontend architecture, tech stack, UX flows, and integration contracts with the existing backend idea.

---

1) Product framing

Primary users

- Inspector: uploads .360 files, monitors job status, views resulting model & report.
- Reviewer: opens models, leaves annotations/notes.
- Company Owner (Admin): full visibility + admin tools: user & role management, sponsor/workspace management, storage/quota views, job queue health, configuration.

Top-level goals

1. Trustworthy uploads on poor networks (resume, verify).
2. Clear job tracking (live status, progress, failures).
3. Fast viewing of models & reports directly in the dashboard.
4. Operational levers for admins without touching infra.

Success metrics (initial)

- Upload completion rate and average time to complete.
- Time from upload → "viewable model".
- % jobs where user can diagnose failure without support.

- Viewer FPS and load time for typical model size.

---

2) Information architecture

- Dashboard
    - Jobs (list, filters: status, date, whiteboard ID, inspector)
    - Create job / Upload (guided flow)
    - Job detail (status timeline, artifacts, model viewer, report preview, logs)
    - Notifications (toasts + inbox for long-running jobs)
- Admin
    - Users & Roles (RBAC, invite, suspend)
    - Sponsors/Workspaces (Drive folders, quotas)
    - Processing Health (Drive agent heartbeat, last poll, queue size, recent failures)
    - Audit Log (who did what, when)
    - Settings (branding, retention hints, feature flags)
- Account (profile, API tokens if exposed, sign out)

---

3) Proposed tech stack

Core (mainstream choices)

- Framework: Next.js 14 (React 18, App Router) - widely adopted, SSR/SSG, file-based routing.

- Language: TypeScript (strict).
- UI library: Material UI (MUI) for a mature, well-documented component set; icons via @mui/icons-material.
- Styling: MUI theme + utility classes via Tailwind CSS (optional, only if helpful for layout). Keeping to MUI alone is fine.
- State & Data: Redux Toolkit (RTK) + RTK Query for API caching/fetching.
- Forms & Validation: react-hook-form + zod.
- 3D/Model: three.js (GLTF/GLB, DRACO, KTX2). Fallback to <model-viewer> where appropriate.
- HTTP: native fetch with an OpenAPI-generated client (from FastAPI schema using openapi-typescript or openapi-generator).
- Uploads: tus-js-client if backend supports TUS; otherwise simple HTTP Content-Range chunking with resume.
- Real-time: SSE for status streams (simpler ops); WebSocket optional for bi-directional admin actions.
- Charts: Recharts.

Quality & DX

- ESLint + Prettier + TypeScript strict.
- Testing: Jest + React Testing Library; Playwright for e2e (upload → view flow).
- Storybook for components.
- Monitoring: Sentry.

Delivery

- CI: GitHub Actions (typecheck, test, build).
- Hosting: Vercel/Netlify or containerized (Nginx) - no vendor-specific APIs required.
- Assets: code-split viewer; stream models from signed URLs.

---

4) Security, auth, and roles

- Auth: FastAPI handles OAuth (e.g., Google). Frontend stores an HTTP-only session cookie; no access tokens in JS.
- Client generation: consume FastAPI's OpenAPI spec to generate a typed client; version the schema per release.
- RBAC: roles inspector, reviewer, owner, optional sponsor_admin (read from /me).
- Network: all requests to FastAPI; artifact access via short-lived signed/proxied URLs. CORS configured centrally on FastAPI.
- Permissions in UI: role-aware components; disable/guard actions.
- Audit: client includes x-client-action metadata for admin actions.

---

5) Upload & job creation UX

Constraints: large .360 files, unstable field networks, resume & verification required.

Flow

1. Select file → preflight → start chunked upload.
2. Sticky progress card: %/ETA, pause/resume, retry, cancel; auto-retry on network loss.
3. Finalize → receive job_id → route to Job Detail with live status via SSE.

4. Background uploads continue via Service Worker; warn on tab close.

Implementation

- Prefer TUS protocol (if enabled server-side) using tus-js-client for standard resume semantics.
- Otherwise use HTTP Content-Range chunking; checkpoints in indexedDB; SHA-256 per chunk + final checksum.
- Resume tokens issued by FastAPI.

Edge cases

- Duplicate file detected by checksum → "use existing upload" prompt.
- Quota warnings and file policy messages surfaced early.

---

6) Jobs list UX

- Layout: table with density toggle; sticky filters; server-side pagination; saved views per user.
- Columns: Job ID, Whiteboard ID, Created, Status, Progress, Inspector, Duration, Actions.
- Filters: status (UPLOADED, OCR_OK, NEEDS_REVIEW, PUSHED_TO_DRIVE, PROCESSING, PROCESSED, FAILED), date range, user, sponsor/workspace.
- Row adornments: colored status pill + live progress via WebSocket.
- Bulk actions (admin): requeue, cancel, export CSV.

---

7) Job detail UX

Sections

- Header: Job ID, status pill, timestamps, owner, retry/cancel (role-gated), copy share link.
- Status timeline: each backend transition with time deltas + error details, retry counts.
- Artifacts:
    - 3D Model Viewer (GLB preferred):
        - controls: orbit, section cut, measure distance, reset view;
        - performance: PMREM, KTX2 textures, optional geometry decimation toggle;
        - fallback download if WebGL is not supported.
    - Report Preview (PDF in inline viewer) with download.
    - Logs (collapsible, streamed if large).
- Notes/Annotations: per-job markdown notes (stored via backend).

Performance targets

- First contentful viewer UI < 2s on broadband; model ready: progressive render in < 6s for median asset.
- Keep viewer bundle under ~250-300KB gzipped (lazy load loaders & controls).

---

8) Admin features (phaseable)

1. User & Role Management: invite by email, reset, deactivate, role switch.
2. Sponsors/Workspaces: link to Drive IDs, show storage, allow admin to refresh Drive oauth/service status.

3. Processing Health:
    - Drive agent heartbeat time, last completed job, queue length (from backend metrics).
    - Quick actions: requeue job, mark failed as acknowledged.
4. Audit Log: searchable list of actions with user, IP, timestamp, target entity.
5. Feature Flags: toggle experimental viewer features, OCR thresholds (read-only if backend-owned).

---

9) API contracts (frontend expectations)

- OpenAPI: Frontend consumes FastAPI's openapi.json to generate a typed client (build step).
- Auth
    - GET /me → { id, roles[], sponsor_ids[] }
- Uploads
    - TUS endpoints if supported (preferred), else:
    - POST /uploads → { upload_id, part_size }
    - PUT /uploads/:id with Content-Range → { acked_bytes }
    - POST /uploads/:id/finalize → { job_id }
- Jobs
    - GET /jobs?… → paginated list
    - GET /jobs/:id → job, transitions, artifact ids

    - Events: GET /jobs/:id/events (SSE) → { status, progress, message }
- Artifacts
    - GET /artifacts/:file_id/url → short-lived URL
    - GET /jobs/:id/logs
- Admin
    - GET /admin/health
    - POST /jobs/:id/retry, POST /jobs/:id/cancel

---

10) Error handling & observability

- Global: error boundary with helpful messaging & issue code.
- Per-request: standardized error payload → toast + inline remediation hints.
- Retries: TanStack Query retry policy for transient 5xx; exponential backoff.
- Telemetry: Sentry captures user id (hashed), job id, route, network meta.
- Operational views: Admin dashboard surfaces recent failures and drill-down links.

---

11) Accessibility & UX polish

- WCAG 2.1 AA targets.
- Keyboard navigation throughout; focus traps in dialogs; skip links.
- Color contrast compliance; status also conveyed via icons/text, not color alone.

- Reduced motion mode for viewer & transitions.

---

12) Performance strategy

- Route-level code splitting; viewer & pdf preview lazy.
- HTTP caching on list endpoints; ETag diffing for incremental reloads.
- Image/model CDN if applicable; stream models with range requests.
- Prefer GLB+KTX2; precompute DRACO server-side when possible.

---

13) Privacy & data handling

- No Drive tokens in the browser; only short-lived artifact URLs.
- PII minimization in telemetry; scrub report contents from client logs.
- Respect sponsor-specific retention settings in UI (badges, warnings).

---

14) Roadmap (frontend)

Phase 0 - Spike

- POC: chunked upload to backend; simple jobs list & detail; load a GLB in three.js from a signed URL.

Phase 1 - Inspector dashboard

- Production upload flow + resume, jobs list & detail + SSE/WS updates, report preview.

Phase 2 - Admin

- Users/Roles, Health, Audit Log, sponsor workspaces.

Phase 3 - Reviewer tools

- Measurements, annotations, share links, CSV export.

---

15) Open questions

- Exact .360 codecs/containers and expected size ranges?
- Canonical model format from Agisoft (OBJ/PLY/GLB)? If multiple, which to standardize for the web?
- Status event cadence & payload shape from backend?
- Artifact retention timeline and download limits per sponsor?
- Any on-prem deployments requiring air-gapped viewer assets?

---

16) Appendix - Component sketch list

- UploadCard, UploadManager, JobTable, StatusPill, JobTimeline, ModelViewer, PdfPreview, ArtifactList, HealthChart, AdminUserTable, AuditLogTable, QuotaBar, shared Dialog/Drawer/Toasts.

Design system tokens: spacing (4/8 grid), radii xl/2xl, elevation scale, semantic colors for statuses.

Summary

This stack emphasizes reliability of uploads, clear state transitions, and performant model viewing while keeping security boundaries (no direct Drive access) and providing admin visibility. It is intentionally modular to phase features as the backend matures.