CollectIQ — Frontend Project Specification

Scope: This specification defines the frontend architecture, coding standards, UI/UX flows, security posture, and delivery criteria for the CollectIQ web application. It is optimized for an authentication‑first, real‑time pricing product with multi‑agent backends on AWS.

# 1. Tech Stack & Principles

• Next.js 14 (App Router), React 18, TypeScript (not strict)

• Styling: Tailwind CSS + shadcn/ui + Radix primitives

• Auth: Amazon Cognito (Hosted UI or Amplify Auth UI), JWT-backed session via secure HTTP-only cookie

• Data: Typed API client (fetch) + Zod for runtime validation; SWR for client caching; edge-safe code paths

• Charts: Recharts (lazy-loaded)

• Testing: Vitest + React Testing Library; Playwright for E2E

• Tooling: ESLint (next/core-web-vitals), Prettier, Husky + lint-staged, Storybook (optional)

• Accessibility: WCAG 2.2 AA; keyboard-first; visible focus; motion-reduced variants

• Performance: Core Web Vitals (LCP < 2.5s, CLS < 0.1, INP < 200ms); 60fps animation target across Safari/Chrome

# 2. Folder Structure

apps/web/  
 app/  
 (public)/  
 auth/  
 upload/ (protected)  
 vault/ (protected)  
 cards/[id]/ (protected)  
 api/ (route handlers if needed)  
 layout.tsx  
 page.tsx  
 components/  
 auth/  
 cards/  
 ui/  
 lib/  
 api.ts (typed API client)  
 schemas.ts (zod schemas mirrored with backend)  
 auth.ts (Cognito helpers)  
 format.ts (formatters)  
 guards.tsx (route/content guards)  
 styles/  
 tests/  
 e2e/

# 3. Environment & Config

Required env vars (examples):

NEXT\_PUBLIC\_REGION=  
NEXT\_PUBLIC\_COGNITO\_USER\_POOL\_ID=  
NEXT\_PUBLIC\_COGNITO\_USER\_POOL\_CLIENT\_ID=  
NEXT\_PUBLIC\_IDENTITY\_POOL\_ID=  
NEXT\_PUBLIC\_API\_BASE=/  
FEATURE\_FLAGS=pricing,authenticity  
SENTRY\_DSN= (optional)

Rules: never expose secrets in client code; prefer httpOnly cookies for session; no tokens in localStorage.

# 4. Auth‑First UX & Routing

• Entry: unauthenticated users are redirected to /auth. After login, redirect to /upload (first‑run) or /vault (returning users).

• Protected routes: /upload, /vault, /cards/[id] require server‑verified session. Client guard renders a spinner while status resolves.

• Session state: short‑lived JWT; silent refresh; session‑expired modal preserves user action and replays after re‑auth.

• Navbar: shows AccountMenu (email/avatar) and Sign out; hides protected links pre‑auth.

# 5. UI Screens & States

5.1 Auth Screens

• Sign in / Create account (Hosted UI or embedded form)  
• Email verification screen  
• Password reset  
• Session expired modal

5.2 Upload & Camera

• UploadDropzone with file type/size guard (jpg, png, heic; ≤ 12 MB)  
• Camera capture on mobile using getUserMedia; permission helper; orientation fixes; iOS Safari compatibility  
• Progress bar for S3 presign → PUT upload; image preview; cancel/retry

5.3 Identify → Authenticity → Valuation

• CandidateList (top‑k)  
• AuthenticityBadge (0–1 with tooltip)  
• ValuationPanel (low/median/high, compsCount, windowDays, confidence)  
• ProblemDetails alert area for RFC 7807 errors

5.4 Vault & Detail

• VaultGrid with filters/sorts (set, rarity, value)  
• PortfolioSummary (total, sparkline)  
• CardDetail (hi‑res image, valuation history, authenticity trend, actions: delete, refresh valuation)

5.5 Empty/Loading/Error States

• Skeletons for image/card grids; optimistic placeholders; actionable errors with retry.

# 6. Design System

Palette: Vault Blue #1A73E8, Holo Cyan #00C6FF, Carbon Gray #1E1E1E; avoid purple/indigo.

Components: Button, Input, Select, Dialog, Sheet, Tabs, Tooltip, Toast, Card (shadcn/ui).

Tokens: spacing, radius (12–16px), shadows (soft), motion (200–250ms base), focus rings (Holo Cyan).

Theming: light/dark via CSS variables; prefers-reduced-motion supported.

Icons: lucide-react; Charts: Recharts (lazy).

# 7. Coding Standards & Patterns

• TypeScript not strict

• Functional components, hooks; no class components.

• Co-locate small hooks with components; share in lib/ for cross‑use.

• API calls centralized in lib/api.ts; never call fetch in components directly.

• Error handling via ProblemDetails type; user‑friendly messages in UI; console logs stripped in prod.

• Accessibility first: semantic HTML, ARIA only when needed, focus management for dialogs/sheets.

• No magic numbers; extract constants; document assumptions near code.

• Safari parity: avoid non‑standard CSS; prefer transform/opacity for animations; throttle heavy JS; use will-change sparingly.

# 8. Typed API Client (lib/api.ts)

Responsibilities: append credentials; handle CSRF (if applicable); parse JSON; validate with Zod; standardize ProblemDetails; auto‑retry idempotent GETs (exponential backoff); no retries on POST/DELETE unless explicitly enabled.

Endpoints:  
POST /api/upload/presign → { key, url }  
POST /api/cards → create/update card  
GET /api/cards → { items[], nextCursor? }  
GET /api/cards/:id → card  
DELETE /api/cards/:id → { ok: true }

# 9. State & Data Fetching

• SWR for caching and revalidation; key per user + route; stale‑while‑revalidate for vault lists.

• Mutations use api client; optimistic UI for add/delete; rollback on failure.

• Suspense boundaries for charts and secondary panels (behind feature flag if needed).

# 10. Image Handling

• Validate MIME (jpeg/png/heic), size, and pixel dimensions before presign. • Use <Image> with placeholder blur; constrain aspect ratios; prevent layout shift. • Generate object URLs for previews; revoke on unmount; compress client‑side selectively (mobile) if needed.

# 11. Accessibility (A11y)

• All interactive elements reachable via keyboard; roving tabindex for grids; focus trap in dialogs; • aria-live for async status (uploading, analyzing); skip links; high contrast tokens; color alone never conveys meaning.

# 12. Performance & Web Vitals

• Budget: LCP < 2.5s (mobile), CLS < 0.1, INP < 200ms. • Code-split routes; lazy-load charts; preconnect to API; cache-control headers. • Prefer CSS transforms; avoid expensive reflows; memoize heavy lists; virtualization if vault > 200 items.

# 13. Security Posture (Frontend)

• Sessions via secure, SameSite=Lax, HTTP-only cookies; no tokens in JS. • Strict CSP (no inline scripts), Referrer-Policy strict-origin-when-cross-origin, X-Frame-Options DENY. • Input validation client-side mirrors Zod schemas; server remains source of truth.

# 14. Analytics & Telemetry

• Event model: upload\_started, upload\_succeeded, analyze\_started, analyze\_succeeded, card\_saved, valuation\_refreshed. • Do not log PII; include requestId when present. • Toggle analytics via env; respect DNT.

# 15. Feature Flags

• FEATURES=pricing,authenticity,charts. • Guard risky features (webcam capture on iOS) behind flags. • Simple boolean gate via env for build; consider server-provided flags later.

# 16. Error Handling & Problem Details

• All API errors shaped as RFC 7807 ProblemDetails: { type, title, status, detail, instance }. • Map common cases: 401 (auth), 403 (ownership), 413 (image too large), 415 (unsupported media), 429 (rate limit), 5xx (retry/backoff). • Provide helpful remediation text; avoid dead ends; always include a Retry action.

# 17. Testing Strategy

Unit (Vitest): formatters, guards, api client, zod parsers, components with logic.

Integration: upload → presign → optimistic UI → finalize; vault list pagination; error states.

E2E (Playwright): auth redirect to /auth; happy path upload→identify→valuation→save; session expiry modal; Safari parity checks.

A11y: axe-core jest checks; keyboard navigation snapshots.

Performance: Lighthouse CI threshold; trace core routes.

# 18. Delivery Workflow

• GitHub Flow; PRs < 300 LOC; reviewers: 1+; status checks: lint, typecheck, test. • Preview deploys per PR (Amplify/Netlify). • Conventional Commits; semantic versioning for UI package if extracted.

# 19. Milestones & Acceptance Criteria

M1 — Auth + Shell: /auth, guard, navbar, route protection. ✅ Criteria: unauth redirect, session badge visible.

M2 — Upload: presign + progress + preview + retry. ✅ Criteria: valid types/sizes; PUT uploads succeed; skeletons shown.

M3 — Identify/Value UI: candidate list, authenticity badge, valuation panel. ✅ Criteria: renders with live API; problem-details handled.

M4 — Vault & Detail: grid, filters, totals, detail view. ✅ Criteria: paginated list; delete/refresh actions; charts lazy loaded.

M5 — A11y & Perf: Lighthouse ≥ 90; keyboard coverage; motion‑reduced paths. ✅ Criteria: audited and documented.

M6 — E2E + Demo Readiness: scripted flow, recording, fallback guidance. ✅ Criteria: stable for judges/users.

# 20. Implementation Notes (Deep Dives)

20.1 AuthGuard: server‑verified session via cookies(); client shows spinner; redirect preserves ?next param.

20.2 Presign Upload: request {filename, contentType}; backend returns {url,key}; PUT file; on success POST /api/cards with key.

20.3 ProblemDetails: convert to typed error; map to Toast/Alert with remediation and retry.

20.4 Safari Parity: prefer CSS transform; avoid backdrop-filter when performance tanks; switch to PNG fallbacks for heavy blurs.

20.5 Charts: lazy import; min 320px height; skeleton while loading; aria-descriptions for values.

# 21. Example Component Contracts

UploadDropzone(props): accept, maxSizeMB, onSelected(files[])

AuthGuard({ children }): handles pending/authenticated/unauthenticated states

ValuationPanel({ value: { low, median, high, compsCount, windowDays, confidence } })

AuthenticityBadge({ score }): rounded pill with tooltip rationale

# 22. Non-Goals

• Offline‑first mode (beyond basic cache)

• Client‑side cryptography or watermarking

• Heavy SSR for charts (client‑only for now)

# 23. Handoff & Documentation

• README with dev setup, env configuration, common scripts. • ADRs for significant decisions (auth storage, upload flow, error model). • Component stories (Storybook) for critical widgets (optional).

# 24. Definition of Done (Frontend)

• Auth‑first UX with protected routes; reliable upload flow; identification and valuation UI; vault and detail screens; a11y/perf budgets met; tests green; CI passing; demo script validated.