Brand Protection Monitor (PoC)

Frontend Architecture & Project Structure — Source of Truth

Document date: 2026-02-01 | Version: 1.0 FINAL

Role executed: InfraGPT — Senior Frontend Architect (Feature-Driven Design / Screaming Architecture)

This document is a single definitive, implementation-ready frontend specification. It is designed to be consumed directly by engineering teams and automated agents without any follow-up clarification.

Hard constraints: English-only, flow-based layout, 36pt margins, no overlaps, tables wrap and repeat headers, anti-iteration (one-shot).

1. Scope, Inputs, and Non-Negotiable Constraints

1.1 Scope (Confirmed)

This Source of Truth covers the frontend only: React + TypeScript + Vite, TanStack Router, Zustand, TanStack Query, Axios, Zod, and Tailwind CSS. Backend implementation details are referenced only as API contracts and behavioral expectations. No backend code or backend directory structure is produced.

Explicit runtime constraints: no authentication/authorization flows in PoC runtime paths; UI is stateless across reloads (no localStorage/sessionStorage persistence).

Design constraints: light theme only; flow-based layout; dense operational data triage UI; no decorative elements.

1.2 Authoritative Inputs (Source Documents)

Artifact	Purpose in this specification	Notes
PRD — Brand Protection Monitor (PoC) — Source of Truth	Functional scope, user stories, screen definitions, API contracts.	Primary business authority.
UI/UX Implementation Source of Truth (FINAL)	Tokens, component library constraints, screen orchestration rules.	Binding UI decisions; no custom one-off components.
Technical Scope & Orchestration Bible (FINAL)	Operational rules, endpoint behaviors, query params, error codes, state machines.	Primary technical authority for frontend contracts.
PostgreSQL Source of Truth — Full Database Design (FINAL)	DB-backed semantics that impact UI behavior (soft delete, dedupe, filtering).	Frontend uses fields as exposed by API; DB referenced for invariants.
Rules_PDF_Generation.txt	PDF layout and completeness rules.	Non-negotiable output constraints.

1.3 Product Surfaces in Scope

Screen ID	Name	Primary intent	Primary API dependencies
SCR-01	Dashboard	Monitor health + metrics + match triage table + filters + export entry point.	GET /monitor/status; GET /matches; GET /export.csv
SCR-02	Keywords Management	Create/list/search/soft-delete keywords driving matching signal.	GET /keywords; POST /keywords; DELETE /keywords/{keyword_id}
SCR-03	Export Modal	Export current filtered matches dataset to CSV via streaming download.	GET /export.csv (with current filters)

2. Resolved Assumptions and Closed Ambiguities

Area	Binding decision	Implementation impact (frontend)
Auth	No runtime authentication/authorization in PoC.	No login screens, no token storage, no protected-route enforcement; design keeps insertion points for future auth.
Persistence	UI does not persist state across reloads.	Filters and dismissed banners stored only in-memory (Zustand) and reset on refresh.
Theme	Light theme only.	Tailwind tokens created only for light theme; no dark-mode branching.
Layout	Flow-based layout; no absolute positioning; wrapping required.	Tables and text must wrap; only fingerprint may be truncated where explicitly required by UI/UX rules.
Data storage	Only matches are stored; non-matching certs are not persisted.	UI must treat /matches as authoritative evidence dataset; no expectation of browsing full CT history.
Deduplication	Unique key = (cert_fingerprint_sha256, matched_keyword, matched_domain); last_seen_at updates on reappearance.	UI must display first_seen_at as immutable and last_seen_at as mutable; 'new' derived from time window/last run.
Keyword lifecycle	Keyword deletion is soft delete; deleted keywords are ignored in filters and exports.	Keyword selector should use only active, non-deleted keywords; old matches remain visible but may reference keywords not currently selectable.
Export semantics	Exports respect current filters; download is direct from server stream.	Export modal shows filter chips; no client-side CSV generation.

Non-goals (Explicit)

• Real-time full-log processing, multi-log support, takedown workflows, notifications, or risk scoring beyond substring matching are out of scope for this PoC.

• UI visual design beyond the binding tokens and component behaviors is out of scope; the UI must be implemented using the specified component set only.

3. Domain Model and Feature-Driven Design

3.1 Core Domains (Screaming Architecture)

The frontend is structured around product domains that 'scream' intent from the folder tree. Each domain is self-contained under /src/features// and exposes a small public API via index.ts. Shared global scope is intentionally minimal.

Feature folder	**Domain responsibility**	**Primary screens**	**Primary endpoints**
features/monitor	Monitor health, status badge, metrics, polling orchestration.	SCR-01	GET /monitor/status
features/matches	Match triage: filters, server pagination, table rendering, highlighting.	SCR-01	GET /matches
features/keywords	Keyword CRUD, validations, dedupe errors, keyword selector dataset.	SCR-02, SCR-01 (filter)	GET/POST/DELETE /keywords
features/export	Export modal orchestration and server-streamed CSV download.	SCR-03 (invoked from SCR-01)	GET /export.csv
features/app-shell	App shell, navigation, layout primitives, route-level composition.	All	N/A (UI composition)

3.2 Global Minimal Surface (Allowed)

Global folders exist only for truly cross-cutting concerns: Axios instance, QueryClient provider, routing root, and low-level UI primitives. All domain logic (API hooks, schemas, pages) lives inside its feature folder.

4. Routing Architecture (TanStack Router)

Routing is file-based and type-safe. Even though authentication is excluded, routes are designed so guards can be added later without refactoring. In PoC, all routes are public.

Primary routes:

- / → redirect to /dashboard
- /dashboard → SCR-01 Dashboard
- /keywords → SCR-02 Keywords Management

Export is a modal within /dashboard and MUST NOT navigate away.

Route	Page component	Data dependencies (TanStack Query)	Key UI states
/dashboard	features/monitor/pages/dashboard-page.tsx	monitorStatusQuery (poll), matchesQuery (debounced filters), keywordsQuery (for filter dropdown)	loading / empty-no-keywords / empty-no-matches / error banner
/keywords	features/keywords/pages/keywords-page.tsx	keywordsQuery; createKeywordMutation; deleteKeywordMutation	inline validation + API error mapping; optimistic UI optional but not required

5. API Contracts → Zod Schemas → Query Hooks

All API responses MUST be runtime-validated using Zod. If validation fails, the UI enters a controlled error state (persistent banner) and provides a retry action. No unvalidated payload may reach rendering logic.

5.1 Endpoint Summary (Frontend View)

Endpoint	Purpose	Query params / body	Success shape (high level)	Error behaviors relevant to UI
GET /monitor/status	Monitor state + last run metrics for dashboard badge and cards.	none	{ state, last_run_at, last_success_at, last_error_code?, last_error_message?, metrics_last_run? }	state=error shows persistent banner; 500 DB_UNAVAILABLE -> banner + retry
GET /keywords	List/search keywords (non-deleted) used by keyword filter and keywords screen.	q?, page?, page_size? (10|25|50)	{ items[], total }	400 INVALID_QUERY -> inline message; 500 DB_ERROR -> banner
POST /keywords	Create keyword with trim + case-insensitive dedupe.	body: { value }	{ keyword_id, value, normalized_value, status, created_at }	400 VALIDATION_ERROR -> inline; 409 DUPLICATE_KEYWORD -> inline; 500 DB_ERROR -> inline + banner
DELETE /keywords/{keyword_id}	Soft delete keyword (excluded from filters/exports).	path param: keyword_id	{ ok: true }	404 NOT_FOUND -> toast; 500 DB_ERROR -> banner
GET /matches	Paginated matches triage list.	page, page_size, keyword, q, issuer, date_from, date_to, new_only, sort	{ items[], total }	400 INVALID_QUERY -> banner; 500 DB_ERROR -> banner; rate-limit possible on heavy search
GET /export.csv	Stream CSV export respecting current filters.	same as /matches	text/csv stream	429 RATE_LIMITED -> modal error state + retry guidance; 500 EXPORT_ERROR before headers -> modal error

5.2 Canonical Query Keys and Refetch Policies

Query / Mutation	Canonical key	Refetch / cache policy	Notes
Monitor status	['monitor-status']	poll every 10s (suggested), refetch on window focus	UI must remain usable even if status is error; display banner but do not block table/export.
Keywords list	['keywords', { q, page, page_size }]	refetch on mutation success; staleTime 0–10s	For filter dropdown, request page_size=50 and q omitted.
Matches list	['matches', { filters... }]	debounce 300ms on filter changes; server-side pagination	Default on load: page=1, page_size=25, sort=first_seen_desc.

Query / Mutation	Canonical key	Refetch / cache policy	Notes
Create keyword (mutation)	N/A	invalidate ['keywords']	Preserve input on error; show inline validation per UI/UX.
Delete keyword (mutation)	N/A	invalidate ['keywords'] and ['matches'] optional	Matches remain; keyword disappears from filter options after delete.

6. UI System: Tokens, Components, and State Behaviors

6.1 Canonical Tokens (Implement Exactly)

Token	Hex	Usage rule
color.primary	#2563EB	Primary CTAs, focus rings, active states
color.success	#16A34A	Positive confirmations only
color.warning	#F59E0B	Non-blocking risk signals
color.error	#DC2626	Errors, destructive actions
surface.page	#F9FAFB	App background
surface.card	#FFFFFF	Cards, tables, modals
text.primary	#111827	Primary text
text.muted	#6B7280	Secondary metadata

6.2 Required Component Library (No One-Off Components)

All screens MUST be composed from the following components. Each component has required states: default, hover, focus, disabled, loading, error (where applicable).

Component	Location	Used by	Key behaviors
App Shell (top bar + navigation)	features/app-shell/components/app-shell.tsx	All routes	Sticky header; product name; nav links; status badge slot.
Status Badge	features/monitor/components/status-badge.tsx	Dashboard header	Renders idle/running/error; ties error state to banner visibility.
Metric Card	features/monitor/components/metric-card.tsx	Dashboard	Shows processed_count, match_count, parse_error_count, cycle_duration_ms (duration_ms).
Data Table (server-side)	features/matches/components/matches-table.tsx	Dashboard	Pagination, sorting, wrap long text; repeat table header on page scroll (UI).
Filter Bar	features/matches/components/filter-bar.tsx	Dashboard	keyword multi-select, date range, text search, new-only toggle; 300ms debounce.
Modal (Export)	features/export/components/export-modal.tsx	Dashboard	Summarize filters (chips); disable button while preparing; handle 429/500.
Toast + Persistent Error Banner	src/components/feedback/*	All	Banner persists until resolved; toast for non-blocking actions (deleted, etc.).
Skeleton Loader	src/components/feedback/skeleton.tsx	Dashboard	Skeleton cards + rows for initial load.

7. Detailed Frontend Project Structure (Exhaustive)

This tree is the canonical /src layout for a React+TS+Vite project using Feature-Driven Design with strict colocation. Every domain feature is self-contained. File and folder naming is kebab-case.

7.1 Top-Level /src Tree

```
src/
  app/
    app.tsx                         // Root component wiring providers + app shell outlet
    main.tsx                        // Vite entry
  routes/
    __root.tsx                      // TanStack Router root route
    index.ts                        // Route tree export (single source)
    dashboard.route.tsx             // /dashboard route -> DashboardPage
    keywords.route.tsx              // /keywords route -> KeywordsPage
    redirect.route.tsx              // / -> redirect to /dashboard
  providers/
    query-client-provider.tsx       // QueryClient + devtools (optional)
    app-providers.tsx               // Composition of providers (Query, Router, etc.)
  lib/
    axios.ts                        // Axios instance; baseURL; timeouts; error normalization
    env.ts                          // Typed env loader (VITE_API_BASE_URL, etc.)
    query-client.ts                 // QueryClient singleton configuration
    url.ts                          // Query param helpers (encode filters deterministically)
  components/
    ui/                             // Low-level primitives (Button, Input, Select, Modal)
      button.tsx
      input.tsx
      select.tsx
      badge.tsx
      modal.tsx
      table.tsx
      toggle.tsx
      date-range.tsx
    feedback/
      error-banner.tsx              // Persistent banner component
      toast.tsx                     // Toast API + container
      skeleton.tsx                  // Skeleton primitives
  features/
    app-shell/
      components/
        app-shell.tsx               // Header + nav + outlet
        app-nav.tsx                 // Nav links (Dashboard / Keywords)
      index.ts
    monitor/
      api/
        use-monitor-status.ts       // TanStack Query hook: GET /monitor/status
      components/
        status-badge.tsx
        metric-card.tsx
        monitor-header.tsx          // Header row (name, status, last_run_at, Export CTA)
        metrics-row.tsx             // 4 metric cards layout
      pages/
        dashboard-page.tsx          // SCR-01 page composition (shell + header + table)
      types/
        monitor-status.schema.ts    // Zod schema for /monitor/status
        monitor.types.ts            // TS inferred types from zod
      index.ts
    keywords/
      api/
        use-keywords.ts             // Query: GET /keywords
        use-create-keyword.ts        // Mutation: POST /keywords
        use-delete-keyword.ts        // Mutation: DELETE /keywords/{id}
      components/
        keyword-form.tsx             // Input + Add button + inline validation
        keywords-table.tsx           // List with delete actions and timestamps
        keyword-status-badge.tsx     // active/inactive rendering (if needed)
      pages/
        keywords-page.tsx            // SCR-02
      types/
        keyword.schema.ts            // Zod schemas for keyword row and API envelopes
        keyword-input.schema.ts      // Zod input schema (trim + length)
        keyword.types.ts
      utils/
        keyword-normalize.ts         // lower(trim(value)) helpers for UI preview
      index.ts
    matches/
      api/
        use-matches.ts               // Query: GET /matches with filter params
      components/
        filter-bar.tsx               // keyword multi-select + date range + q + new_only
        matches-table.tsx            // Table + pagination + sorting
        match-highlight.tsx          // Safe highlighting (no unsafe HTML)
        match-row-expanded.tsx       // Optional expandable details
      types/
        match.schema.ts              // Zod schema for match rows + list response
        match-filters.schema.ts      // Zod schema for filter state (client-side)
        match.types.ts
      utils/
        build-matches-query.ts       // Canonical query param builder
        highlight.ts                 // String range highlight logic (safe)
      index.ts
    export/
      components/
        export-modal.tsx             // SCR-03; filter chips summary + download CTA
        export-filter-chips.tsx
      utils/
        build-export-url.ts          // Produces /export.csv?{params}
      index.ts
  stores/
    use-ui-store.ts                  // Global ephemeral UI state (banner dismissed, modal open)
  styles/
    tailwind.css                     // Tailwind base entry
    tokens.css                       // CSS variables for canonical tokens
  types/
    api-error.schema.ts              // Zod schema for standard JSON error envelope (if provided)
    pagination.types.ts              // shared pagination types
```

7.2 Feature Folder Contract (Enforced)

Every feature folder follows the same internal contract:

```
/api        // Axios calls + TanStack Query hooks for this domain
/components // Domain-scoped components only used within this feature
/pages      // Route-level screens (TanStack Router targets)
/types      // Zod schemas + inferred TypeScript types
/utils      // Small domain helpers (pure functions)
index.ts    // Barrel: ONLY exports the public surface of the feature
```

8. Implementation Rules (Frontend)

Rule	Definition	Enforcement point
Zod-first contracts	All API responses are parsed via Zod before use.	Query hook layer (api/).
No one-off UI	Use only the defined component library; compose, do not invent.	Code review + lint rule optional.
Safe highlighting	Never inject HTML for highlights; use string slicing and spans.	matches/utils/highlight.ts + match-highlight.tsx.
Server-driven pagination	No client-side filtering of large datasets; always use query params.	matches/api/use-matches.ts.
Debounced filter refetch	Debounce 300ms on filters; do not spam API.	Filter bar hook + query key update.
Ephemeral UI store	Zustand is in-memory only; no persistence plugin.	stores/use-ui-store.ts.
Export is server-streamed	Browser downloads from /export.csv; no client assembly.	export/utils/build-export-url.ts.
Error UX consistency	Persistent banner for critical failures; inline errors for form validation.	components/feedback/error-banner.tsx + feature forms.

9. Edge Cases and Stress Tests (UI)

Scenario	Expected UI behavior	Implementation notes
No keywords configured	Dashboard shows empty state with CTA to Keywords screen; matches query may still run but results expected empty.	Prefer to short-circuit by not enabling matches query until keywords query returns non-empty.
No matches yet	Dashboard shows educational empty state (tips for keywords).	Render table region as empty state; keep filters visible or disabled.
Monitor status = error but DB is healthy	Show persistent error banner with last_error_code/message; table and keyword CRUD remain usable.	Do not block interactions; banner is informational + retry.
Invalid query params (400)	Show banner or inline message depending on surface; reset invalid params to defaults.	Centralize param parsing + defaults in build-matches-query.
Export rate limited (429)	Export modal shows error state and guidance; CTA disabled for cooldown period if provided.	If server provides Retry-After header, honor it.
Very long domains/issuer strings	Wrap text; allow copy; no truncation (except fingerprint if explicitly allowed).	Use CSS: break-words; monospace for fingerprint.
High volume (10k+ matches)	Table remains responsive with server pagination; loading states on page changes.	Keep row rendering simple; avoid expensive highlight on huge strings.
API returns unexpected shape	Controlled error state; do not render partial corrupted UI.	Zod parse failure triggers banner + logs.

10. Decision Record (Key Architectural Choices)

Decision	Chosen option	Rationale
Feature-Driven Design with strict colocation	All domain logic under /src/features/*	Scales to many routes; reduces cross-feature coupling; makes ownership obvious.
TanStack Query for server state	Query hooks per feature, canonical keys	Handles caching, refetch, retries; clean separation of fetch and UI.
Zod runtime validation	Parse all responses and inputs	Prevents UI crashes on contract drift; enables safe fallbacks.
Minimal Zustand usage	Ephemeral UI flags only	PoC constraint: no persistence; keeps mental model simple.
Server-driven filtering + pagination	All filters map to query params	Meets NFR performance constraints; avoids client-side heavy work.
Export as direct download	No client CSV generation	Streaming requirement; avoids memory spikes; respects server filter snapshot.
No auth in PoC	Design insertion points only	Aligned with PoC constraints while remaining extensible.

11. Traceability Matrix (Requirements → UI → Feature Ownership)

Requirement / Story	UI surface	Feature owner	Primary artifacts / endpoints
HU-KW-01 Create keyword	SCR-02	features/keywords	POST /keywords; keywords table (soft-delete semantics)
HU-KW-02 Delete keyword (soft)	SCR-02	features/keywords	DELETE /keywords/{id}; keyword removed from filters
HU-KW-03 List/search keywords	SCR-02 + SCR-01 filter	features/keywords	GET /keywords?q=&page;=&page;_size=
HU-DASH-01 Monitor status	SCR-01 header + cards + banner	features/monitor	GET /monitor/status
HU-MAT-01 List matches	SCR-01 table	features/matches	GET /matches (paginated)
HU-MAT-02 Filter/sort matches	SCR-01 filter bar	features/matches	GET /matches with keyword/q/issuer/date/new_only/sort
HU-EXP-01 Export CSV	SCR-03 modal	features/export	GET /export.csv with same params as /matches
HU-SEC-01 Input validation	SCR-02 keyword form	features/keywords	Client-side zod + server validation; show errors
HU-SEC-02 Rate limiting resilience	SCR-03 modal + filter/search UX	features/export + features/matches	Handle 429/400 gracefully; debounce filters

12. Evolution and Scaling Strategy

The PoC frontend is designed to evolve without structural refactors:

• Authentication insertion: add auth provider + route guards in /src/providers and /src/routes without modifying feature internals.

• Multi-log support: introduce a new feature (features/logs) and add log selector to filter bar; extend query param builder in matches/export.

• Risk scoring and triage workflow: new feature (features/triage) for labeling, notes, and relevance; keep matches read model intact.

• Reporting and aggregates: new feature (features/reports) consuming new endpoints; no impact to monitor/matches contracts.

• Design system hardening: migrate /components/ui to shadcn-based primitives while keeping feature components stable.

13. Final Acceptance Checklist (Completeness + PDF Rules)

Category	Checklist item	Status (must be YES)
Content completeness	Frontend-only scope fully specified; no missing screens or domains.	YES
Content completeness	All binding assumptions and ambiguities are explicitly closed.	YES
Traceability	Each screen/user story mapped to feature ownership and endpoints.	YES
Implementation readiness	Exhaustive /src tree with responsibilities and colocation rules.	YES
Implementation readiness	API contracts translated into Zod-first + Query hook patterns.	YES
Edge cases	Stress scenarios defined with expected UI behaviors.	YES
PDF layout rules	Flow-based layout only; no absolute positioning.	YES
PDF layout rules	36pt margins on all sides; no content outside printable area.	YES
PDF layout rules	Tables wrap; top-aligned; repeat header rows across pages.	YES
PDF layout rules	No text truncation (except fingerprint when explicitly allowed).	YES

End of document — SINGLE definitive generation.