Smart-Query Platform — Today's Work

Date: April 17, 2026 **Built by:** Jeff Franzen (Dragon) · with Claude Opus 4.7

The mental model (read first)

Every app you touched is a **window into the same brain** (`explorer.jbf.com/api/smart-query`). Change how the brain thinks → every window inherits it. The four apps just pre-seed different contexts ("you're a DOC officer" on sitaware, "user is viewing FL" on ops, etc.).

The brain knows four things:

1. **Federal data** (county_rankings, fema_declarations, alice, svi, nri) — for rankings and numbers
2. **20,686 ingested docs** — for narrative
3. **Wikipedia FTS** — for exact terms
4. **FLARE live layer** — for fire response gaps

Each data source was added because it answered a class of question nothing else could. That discipline is what keeps the system from bloating.

---

1. Supabase project `qoskpyfgimjcmmxunfji`

What we did:

- Created `execute_read_sql` RPC — a Postgres function that safely runs user-supplied SELECT queries
- Created `wiki_counties` table with 3,144 rows (real Wikipedia content for every US county)
- Added GIN tsvector index for BM25-style full-text search
- Created `search_wiki_counties` RPC (ranked full-text search with highlighted snippets)
- Tightened RLS after the bulk load so the anon key is read-only again

How it works: Your anon key + these RPCs let any of your frontends run safe, scoped SQL + full-text searches over 3,144 counties without needing a server. PostgREST enforces read-only via the RPC validation logic.

---

2. `dragons-brain` / LightRAG (Railway)

What we did:

- Ingested 12,834 Supabase-sourced documents (hazard profiles, FEMA declarations, SVI aggregated, NRI aggregated)
- Scraped 3,110 US county Wikipedia articles and ingested those
- New scripts: `ingest_supabase.py`, `ingest_wikipedia.py`, `populate_wiki_supabase.py`
- Final corpus: **20,686 processed documents**

How it works: LightRAG runs on Railway. When you POST to `/query`, it (a) searches vector embeddings, (b) traverses a Neo4j entity-relationship graph, (c) sends relevant chunks to GPT-4o-mini for synthesis, (d) returns a narrative answer with citations.

---

3. `vulnerability-explorer` → `explorer.jbf.com` (the brain)

What we did: Built `/api/smart-query` — the central reasoning engine — and wired its UI.

Four retrieval tools Claude can call

Tool	What it does	When Claude picks it
`supabase_sql`	SELECT queries via the RPC	Rankings, counts, computed totals
`lightrag_query`	Semantic RAG over 20,686 docs	Narrative, explanations, cross-doc synthesis
`text_search`	Postgres tsvector+GIN FTS	Exact-token matches ("Tlingit", "Treasure Coast")
`agol_query`	Live AGOL layer calls	FLARE fire incidents (RC response gaps)

Plus: auto-enrich county answers with chapter/region/division, strip lat/lon for privacy, honest "I don't know" detection, sample questions regrouped by Red Cross persona.

How it works: User types a question → Vercel serverless function → Claude Opus 4.7 agentic loop → Claude decides which tool(s) to use → executes them → synthesizes a markdown answer → returns. Every answer includes a `trace` of what tools ran.

---

4. `spatial-ops-jbf` ⟶ `ops.jbf.com`

What we did:

- Added `ANTHROPIC_API_KEY` env var (was missing — natural query was broken)
- Fixed the 3-day parcel filter bug (switched from paint-property `case` expressions to layer `setFilter`)
- Replaced the broken QUERY tab backend with a smart-query proxy + markdown rendering
- Made the right panel resizable (drag left edge, saved to localStorage)

How it works: QUERY tab sends questions to `explorer.jbf.com/api/smart-query`, gets a markdown answer, renders it via `marked.js` inside the existing results panel. The current state (FL, TX, etc.) is passed as system-prompt context.

---

5. `county-intel-v2` → `county.jbf.com`

What we did:

- Added `LIGHTRAG_URL` / `LIGHTRAG_KEY` env vars (was 500-erroring)
- Rewrote the narrative panel to call smart-query (fixed the "LA County shows Pinellas" bug)
- Added a follow-up chat input — ask more questions about the same county
- Added a free-form question fallback — typing anything weird no longer crashes the panels

How it works: Search a county → structured panel loads (real SQL) + map loads + smart-query brief loads in the narrative panel. User can chat follow-ups; each gets scoped to the current county automatically.

---

6. `sitaware` → `sitaware.jbf.com`

What we did:

- Added a floating "?" Ask drawer with dark-theme styling
- DOC-officer tactical system prompt context
- **Gated behind `?ask=1` URL flag, `Cmd/Ctrl-K` keyboard shortcut, or `askEnable()` console command** — invisible to general DOC officers until deliberately turned on

How it works: Power users enable it once (persists in localStorage). After that, the red "?" FAB appears bottom-right. Click opens a drawer, questions go to smart-query with DOC tactical context pre-seeded. Drawer auto-hides the FAB while open so there's no collision.

---

7. `flare-v2` → `flare-v2-red.vercel.app` (sourced, not modified)

What we did: Discovered and verified its data sources. The FLARE Fire Incidents CY2024 AGOL layer (103,400 real incidents) is publicly accessible without AGOL login — safe for smart-query.

What we did NOT touch: Flare's own UI. It keeps using its own internal render logic with 500m client-side jitter.

---

8. Obsidian vault (`~/dev/dragons-brain-vault`)

- Session note `sessions/2026-04-16.md` — yesterday's LightRAG ingestion
- Updated `projects/dragons-brain-lightrag.md` with final numbers
- **NEW:** **`projects/smart-query-platform.md`** — the strategic plan (printed companion to this document)
- Memory files: Supabase tables inventory, LightRAG ingestion reference, parcel filter feedback
- Archived stale iCloud-orphaned vault copy (the paid Obsidian Sync vault at `~/dev` is now the single source of truth)

---

9. Commits status

- `vulnerability-explorer-deploy` — committed (hash `caa801c`)
- `spatial-ops-jbf` — **uncommitted** (live site has the changes; git record pending)
- `county-intel-v2` — **uncommitted**
- `sitaware` — **uncommitted**
- `dragons-brain` — **uncommitted** (new ingestion scripts + Wikipedia work)

Nothing pushed to GitHub yet. Deploys to Vercel happened separately and are live regardless of git state.

Key demonstrations

These are the moments that proved the architecture works. Each was run today in the live system:

1. **"What are the five most populous counties with the highest ALICE household counts?"**
2. LA 1,530,874 · Harris 779,823 · Cook 706,552 · Maricopa 587,705 · Miami-Dade 501,954
3. Exact SQL. Compare to ChatGPT's fabricated Polk County IA at 50K.
4. **"Which 10 chapters had the most fires without RC notification in 2024?"**
5. ARC of Greater Chicago 1,154 · ARC of Southeast Michigan 940 · ARC serving the Northern Valleys 921 · (…)
6. Real FLARE incident data, service-delivery gap map no one at Red Cross had exposed this plainly.
7. **"Which Florida county has the highest hurricane risk, and what is that county known for?"**
8. Indian River County (98.63) with geographic/economic narrative from LightRAG + Wikipedia
9. Hybrid query using both `supabase_sql` (ranking) and `lightrag_query` (narrative) in sequence.
10. **"Which Alaska counties mention Tlingit culture?"**
11. 6 ranked results with highlighted snippets via BM25 full-text search
12. The `text_search` tool catching what vector search would miss.

Privacy posture

Every FLARE answer strips `Latitude` / `Longitude` before Claude sees the data. Belt-and-suspenders: both LLM-layer directive (refuse to promise coordinates) and server-layer strip (never returns them in response rows). Users wanting mapped point data are directed to the Flare app where 500m client-side jitter is applied at render time.

Companion document: see `Smart-Query Platform — Strategic Plan.html` for the roadmap.