Project Structure — CompIntelMon: Agentic Competitive Intelligence Monitoring Platform

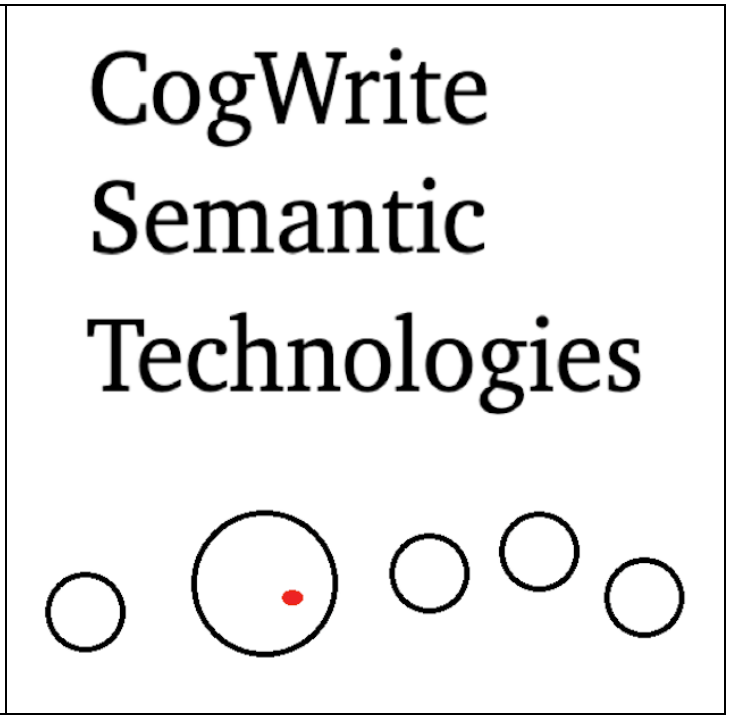


Table of Contents

1. Overview

CompIntelMon is a multi-tenant competitive intelligence monitoring platform. It uses CrewAI agents backed by Anthropic Claude to automatically discover, collect, analyze, and report on competitive data from web sources. Users define subjects to monitor (companies, products, services, topics), and the platform handles the intelligence cycle: source discovery → data collection → analysis → report generation.

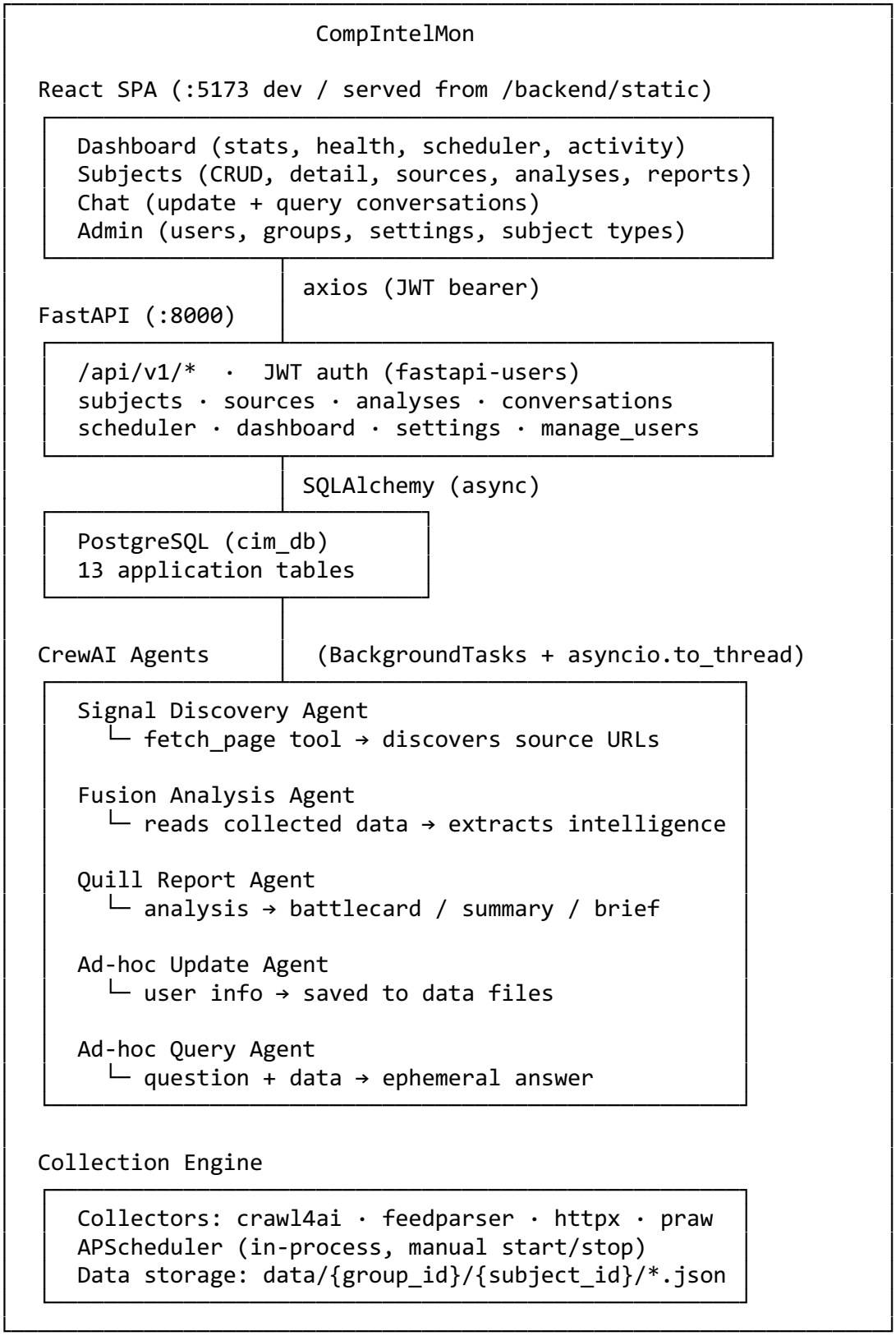
The platform has two runtime components:

1. **FastAPI web server** (*backend/main.py*) — serves the React SPA, all REST API endpoints, and runs the APScheduler for periodic collection
2. **React + TypeScript frontend** — Bootstrap-based UI with subject management, chat interface, and real-time status polling

Version: 0.9.0 (2026-04-04) — built in 9 phases over two days (2026-04-01 to 2026-04-02).

Database: PostgreSQL (*cim_db*), test database (*cim_db_test*).

2. System Architecture



3. Tech Stack

A. Backend

Layer	Technology	Version
Language	Python	3.12
Web framework	FastAPI	0.135+
ASGI server	Uvicorn	0.42+
Auth	fastapi-users (JWT)	15.0+
ORM	SQLAlchemy (async + sync)	2.0.48+
Async DB driver	asyncpg	0.31+
Sync DB driver	psycopg2-binary	2.9+
Migrations	Alembic	1.18+
Agent framework	CrewAI (with Anthropic)	1.12+
LLM provider	Anthropic Claude (Sonnet)	via LiteLLM
Web scraping	crawl4ai	0.8+
RSS parsing	feedparser	6.0+
Reddit API	praw	7.7+
Scheduling	APScheduler (AsyncIOScheduler)	3.10+
Logging	structlog (JSON)	25.5+
HTTP client	httpx	0.28+

1. Frontend

Layer	Technology	Version
Framework	React	19.1
Language	TypeScript	5.8
Build tool	Vite	7.1+
UI library	React Bootstrap	2.10
Icons	react-bootstrap-icons	1.11+
State management	Zustand	5.0
HTTP client	Axios	1.12
Routing	react-router-dom	7.8
Markdown	react-markdown	10.1+

4. Directory Layout

```
compintelmon_code/
├── backend/
│   ├── main.py                    # Entry point: DB wait, seed, app + routers, SPA
│   ├── app.py                     # FastAPI factory (lifespan, CORS, middleware)
│   ├── config.py                  # .env loading, all config constants
│   ├── requirements.txt           # Python dependencies (18 packages)
│   ├── .env                       # Environment config (not committed)
│   │
│   ├── agents/                    # CrewAI agent definitions
│   │   ├── signal_discovery.py    #   URL discovery agent (fetch_page tool)
│   │   ├── fusion_analysis.py     #   Competitive intelligence analysis agent
│   │   ├── quill_report.py        #   Report generation agent (battlecard/summary)
│   │   ├── adhoc_update.py        #   Data update processing agent
│   │   └── adhoc_query.py         #   Ephemeral Q&A agent
│   │
│   ├── api/                       # FastAPI routers (11 modules)
│   │   ├── __init__.py            #   Route aggregation (api_router)
│   │   ├── subjects.py            #   Subject CRUD + auto-provisioning
│   │   ├── sources.py             #   Playbook templates + source CRUD + collection
│   │   ├── analyses.py            #   Analysis + report endpoints
│   │   ├── conversations.py       #   Chat conversations + messages
│   │   ├── dashboard.py           #   Stats, recent runs, system health
│   │   ├── scheduler.py           #   Start/stop/status scheduler
│   │   ├── settings.py            #   Global + group settings, webapp_options
│   │   ├── groups.py              #   Group CRUD (superuser)
│   │   ├── manage_users.py        #   User CRUD with role assignment
│   │   ├── subject_types.py       #   Subject type CRUD (superuser)
│   │   └── users.py               #   /users/me endpoint
│   │
│   ├── auth/                      # Authentication
│   │   ├── users.py               #   fastapi-users config (JWT, UserManager,
│   │   │                          #     username+email login, group active check)
│   │   └── middleware.py          #   refresh_last_seen (60s debounce)
│   │
│   ├── collectors/                # Data collection adapters
│   │   ├── base.py                #   CollectionResult, registry, interpolation
│   │   ├── crawl4ai_collector.py  #   Web page scraping (async, Chromium)
│   │   ├── feedparser_collector.py#   RSS/Atom feed parsing
│   │   ├── httpx_collector.py     #   HTTP URL fetching
│   │   └── praw_collector.py      #   Reddit search via PRAW
│   │
│   ├── db/                        # Database layer
│   │   ├── __init__.py            #   Base + model imports
│   │   ├── models.py              #   13 SQLAlchemy models
│   │   ├── schemas.py             #   Pydantic request/response schemas (~40)
│   │   ├── session.py             #   Async/sync engine, session factory
│   │   ├── seed.py                #   Idempotent startup seeder
│   │   ├── playbook_defaults.py   #   55 default playbook template records
```

```
│   │   │
│   │   └── tables/                    #   Repository-pattern DB access (11 classes)
│   │       ├── analyses.py
│   │       ├── api_groups.py
│   │       ├── api_settings.py
│   │       ├── conversation_messages.py
│   │       ├── conversations.py
│   │       ├── group_settings.py
│   │       ├── group_subjects.py
│   │       ├── playbook_templates.py
│   │       ├── subject_source_runs.py
│   │       ├── subject_sources.py
│   │       ├── subject_types.py
│   │       └── reports.py
│   │
│   ├── services/                      # Background task orchestrators
│   │   ├── collection_runner.py       #   Source collection (dispatch to collectors)
│   │   ├── discovery_runner.py        #   Signal agent URL discovery
│   │   ├── analysis_runner.py         #   Fusion analysis orchestration
│   │   ├── report_runner.py           #   Quill report generation
│   │   ├── chat_runner.py             #   Ad-hoc update/query processing
│   │   ├── scheduler_service.py       #   APScheduler singleton (start/stop/dispatch)
│   │   ├── data_utils.py              #   Shared file I/O utilities
│   │   └── logger_service.py          #   structlog JSON configuration
│   │
│   ├── tests/                         # pytest test suite (12 modules, 89 tests)
│   │   ├── conftest.py                #   Fixtures, test DB, NullPool, seeding
│   │   ├── test_auth.py               #   Login tests (4)
│   │   ├── test_users.py              #   User endpoint tests (3)
│   │   ├── test_settings.py           #   Settings CRUD tests (4)
│   │   ├── test_subjects.py           #   Subject CRUD tests (3)
│   │   ├── test_playbooks.py          #   Template/source provisioning tests (9)
│   │   ├── test_collection.py         #   Collector + collection API tests (16)
│   │   ├── test_discovery.py          #   Discovery agent tests (9)
│   │   ├── test_scheduler.py          #   Scheduler + dashboard tests (8)
│   │   ├── test_analyses.py           #   Analysis + report tests (8)
│   │   ├── test_conversations.py      #   Chat conversation tests (7)
│   │   ├── test_subject_types.py      #   Subject type CRUD + clone tests (8)
│   │   └── test_manage.py             #   Group/user management tests (10)
│   │
│   ├── alembic/                       # Database migrations (6 versions)
│   │   ├── env.py
│   │   ├── script.py.mako
│   │   └── versions/
│   │
│   ├── landing/                       # Public landing page
│   │   ├── index.html
│   │   └── images/                    #   CogWrite branding images
│   │
│   └── static/                        # Compiled frontend assets (built by Vite)
│
├── frontend/                          # React + TypeScript SPA
```

```
│   ├── index.html
│   ├── package.json
│   ├── tsconfig.json
│   ├── vite.config.ts
│   ├── public/                          #   Favicon files
│   │
│   └── src/
│       ├── main.tsx                     #   React entry point
│       ├── App.tsx                      #   Root layout (auth check, TopNavBar, SideMenu)
│       ├── Router.tsx                   #   Route definitions
│       ├── theme.css                    #   Theme color CSS variables + utility classes
│       │
│       ├── api/                         #   API client utilities
│       │   ├── axiosClient.ts           #     Axios with JWT interceptor + 401 handler
│       │   ├── apiURL.ts                #     API_URL constant
│       │   ├── NavigationInjector.tsx   #      Unauthenticated redirect callback
│       │   └── getColor.ts              #     21-color palette (Tailwind shades 50-900)
│       │
│       ├── stores/                      #   Zustand state stores
│       │   ├── useAuthStore.ts          #     User session (roles, group)
│       │   └── useSettingsStore.ts      #     App settings + theme color injection
│       │
│       ├── components/                  #   Reusable UI components
│       │   ├── TopNavBar.tsx            #     Themed navbar (bg-tc-300)
│       │   ├── SideMenu.tsx             #     Left sidebar (role-conditional)
│       │   ├── ProtectedRoute.tsx       #     Role-gated route wrapper
│       │   ├── SubjectFormModal.tsx     #     Subject create/edit (dynamic type dropdown)
│       │   └── ConfirmDeleteModal.tsx   #     Generic delete confirmation
│       │
│       └── pages/                       #   Page components
│           ├── Dashboard.tsx            #     Stats cards, scheduler, health, activity
│           ├── Login.tsx                #     Username/email + password form
│           ├── Logout.tsx               #     Token clear + redirect
│           ├── Subjects.tsx             #     Subject list with CRUD
│           ├── SubjectDetail.tsx        #     Sources, analyses, reports, actions
│           ├── SubjectChat.tsx          #     Update/Query chat interface
│           ├── Settings.tsx             #     Global settings CRUD (superuser)
│           ├── GroupSettings.tsx        #     Group settings CRUD (groupadmin+)
│           ├── ManageUsers.tsx          #     User management with roles
│           ├── ManageGroups.tsx         #     Group management with active toggle
│           └── AdminSubjectTypes.tsx    #     Subject types + template CRUD/clone
│
├── data/                                # Collected data (file-based storage)
│   └── {group_id}/
│       └── {subject_id}/
│           ├── {date}_{source}_{run}.json    # Collection output
│           ├── {date}_adhoc_update_{msg}.json # Chat update data
│           └── reports/
│               └── {date}_{type}_{id}.md     # Markdown reports
│
├── alembic.ini                          # Alembic configuration
├── .env                                 # Environment variables (not committed)
```

```
├── .env.example                    # Environment template
└── .gitignore
```

---

5. Multi-Tenancy Architecture

2. Tenant Isolation

CompIntelMon uses **row-level multi-tenancy** — all tenant-specific tables carry a *group_id* foreign key, and all queries filter by it.

Level	Isolation Strategy
Database	All queries filter by *group_id*; FK constraints enforce relationships
Subjects	Subjects belong to groups; users see only their group's subjects
Sources	Sources belong to subjects, which belong to groups
Collected data	Files stored in *data/{group_id}/{subject_id}/* directories
Settings	*group_settings* table provides per-group configuration
Application	User→Group mapping via *api_users.group_id*; all routes check group

3. User Roles (Hierarchical)

Role	Flag	Capabilities
User	(all flags false)	View reports, view status, issue ad-hoc queries against group subjects
Subject Manager	*is_subjectmanager*	+ edit workflow frequency, start workflows, create subjects
Group Admin	*is_groupadmin*	+ manage users in their group, assign user/subjectmanager/groupadmin roles, manage group settings
Superuser	*is_superuser*	Full system access: all groups, all users, global settings, subject types, playbook templates. Only a superuser can create another superuser

Login requires both the user's *is_active* flag AND their group's *is_active* flag to be true (superusers bypass the group check).

---

6. Database — *cim_db*

PostgreSQL 17.x, 13 application tables.

4. Core Tables

Table	Purpose
subject_types	Dynamic subject type definitions (company, product, service, topic + user-defined)
group_subjects	Monitored subjects: name, type (FK), status, enabled
playbook_templates	Source collection templates: 55 defaults across 4 types, fully editable, clonable
subject_sources	Per-subject source configurations (provisioned from templates): tool, frequency, user_inputs, status
subject_source_runs	Collection run audit log: started_at, status, items_collected, data_hash
analyses	Fusion agent output: summary, key_findings (JSON), signals (JSON)
reports	Quill agent output: title, content_markdown, report_type
conversations	Chat threads: gsubject_id, user_id, type (update/query)
conversation_messages	Chat messages: role, content, status, metadata

5. Tenant Tables

Table	Purpose
api_groups	Tenant groups with *is_active* flag
api_users	Users (extends fastapi-users UUID model): email, username, group_id, role flags, last_seen
api_settings	Key-value global settings: app_title, navbar_color, instance_label, dashboard_*, versions
group_settings	Per-group key-value settings: API keys (Reddit), feature toggles (markdown reports)

7. Intelligence Pipeline

6. Phase 1: Source Discovery (Signal Agent)

When a subject is created, 55 source templates are evaluated and matching ones provisioned. The Signal Discovery Agent (CrewAI) then:

1. Takes the subject name + type
2. Uses *fetch_page* tool to visit the main website
3. Discovers URLs for blog, careers, pricing, docs, changelog, status page, social media
4. Populates *user_inputs* on each source with discovered URLs

7.

8. Phase 2: Data Collection

Collectors fetch data from configured sources:

Collector	Tool	Sources
crawl4ai	AsyncWebCrawler (Chromium)	Websites, blogs, careers, docs, reviews
feedparser	RSS/Atom parser	Blog feeds, press releases, newsletters
httpx	HTTP client	Status pages, API endpoints
praw	Reddit API (PRAW)	Reddit mentions, discussions

Collection can be triggered manually (Collect All / per-source play button), or automatically via APScheduler based on each source's *frequency_minutes*.

Collected data is saved as JSON files: *data/{group}/{subject}/{date}_{source}_{run}.json*

9. Phase 3: Analysis (Fusion Agent)

On-demand analysis triggered by "Analyze" button:

1. Loads latest collected data for each source
2. Passes raw content + signal_instructions to Fusion Agent
3. Agent extracts: summary, key_findings[], signals[]
4. Results saved to *analyses* table

10. Phase 4: Report Generation (Quill Agent)

On-demand report from a completed analysis:

1. Loads analysis (summary, findings, signals)
2. Quill Agent generates formatted Markdown report
3. Report types: battlecard, summary, executive_brief
4. Saved to *reports* table + optional markdown file

11. Phase 5: Ad-hoc Interaction (Chat)

Two conversation modes per subject:

- **Update**: User provides new information or directs investigation → data saved for future analyses
- **Query**: User asks questions against collected data → ephemeral answer, nothing saved

---

8. Playbook Template System

12. Template Architecture

55 default templates organized by subject type:

Subject Type	Templates	Default Enabled
Company	19	5
Product	12	5
Service	12	8
Topic	12	0

13. Template Readiness

Status	Count	Description
Working	21	Fully operational (crawl4ai/feedparser/httpx with URL templates)
Working if URL provided	4	Need user/discovery to provide URL
Needs search collector	19	Require dedicated API integrations (news, GitHub, patents)
Needs API key	6	Twitter (tweepy) / Reddit (praw) — need credentials
Needs playwright	3	JS-rendered pricing pages
Fragile scraping	1	Glassdoor — works but may get blocked
Scraping restricted	1	LinkedIn — ToS issues

14. Template Customization

Superusers can: - Create new subject types - Add/edit/delete playbook templates per type - Clone templates between subject types - All via the Admin Subject Types page

---

9. API Routes

Base prefix: `/api/v1`

15. Authentication (fastapi-users)

Method	Path	Description
POST	`/auth/jwt/login`	Login by email or username → JWT token (7-day expiry)
POST	`/auth/register`	Register new user
GET	`/users/me`	Current user info (includes group_name, role flags)

16. Subjects (authenticated)

Method	Path	Description
GET	`/subjects`	List subjects for user's group
POST	`/subjects`	Create subject + auto-provision sources from templates
GET/PUT/DELETE	`/subjects/{id}`	Read, update, soft-delete subject

17. Sources & Collection

Method	Path	Description
GET	`/playbook-templates`	List templates (filter by subject_type)
POST/PUT	`/playbook-templates[/{id}]`	Create/update template (superuser)
POST	`/playbook-templates/{id}/clone`	Clone template to another type
GET	`/subjects/{id}/sources`	List sources for subject
PUT/POST/DELETE	`/subjects/{id}/sources[/{sid}]`	CRUD sources
POST	`/subjects/{id}/sources/{sid}/collect`	Trigger single source collection
POST	`/subjects/{id}/collect-all`	Trigger all enabled sources
POST	`/subjects/{id}/discover`	Run Signal Discovery Agent
GET	`/subjects/{id}/sources/{sid}/runs`	Collection run history

18. Analysis & Reports

Method	Path	Description
POST	`/subjects/{id}/analyze`	Trigger Fusion analysis
GET	`/subjects/{id}/analyses`	List analyses
POST	`/subjects/{id}/analyses/{aid}/report`	Generate Quill report
GET	`/subjects/{id}/reports`	List reports

19. Chat Conversations

Method	Path	Description
POST	`/subjects/{id}/conversations`	Create conversation (update/query)
GET	`/subjects/{id}/conversations`	List conversations
POST	`.../conversations/{cid}/messages`	Send message (triggers background agent)
GET	`.../conversations/{cid}/messages`	Poll messages (after_message_id filtering)

20. Dashboard & Scheduler

Method	Path	Description
GET	`/dashboard/stats`	Subject count, source count, due count, scheduler status
GET	`/dashboard/recent-runs`	Last N collection runs with subject/source names
GET	`/dashboard/health`	System health checks (PostgreSQL, API key, Chromium, Reddit, disk)
POST	`/scheduler/start`	Start periodic collection (superuser)
POST	`/scheduler/stop`	Stop scheduler (superuser)
GET	`/scheduler/status`	Running/stopped + next check time

21. Administration

Method	Path	Description
GET	`/subject-types`	List subject types
POST/PUT	`/subject-types[/{id}]`	Create/update type (superuser)
GET/PUT	`/settings`	Global settings CRUD (superuser)
GET/PUT/DELETE	`/group_settings/{gid}[/{name}]`	Group settings CRUD (groupadmin+)
GET/POST/PUT	`/groups[/{id}]`	Group CRUD (superuser)
GET/POST/PUT	`/manage/users[/{id}]`	User CRUD with role assignment

22. Public

Method	Path	Description
GET	*`/webapp_options`*	Theme, titles, version (no auth required)
GET	*/*	Landing page (CogWrite branding)

---

10. Frontend Pages

23. URL Routes

Path	Page	Access
/	Landing page (CogWrite branding, login link)	Public
/app/login	Login form	Public
/app	Dashboard (stats, health, scheduler, activity)	Auth
/app/subjects	Subject list with CRUD	Auth
/app/subjects/:id	Subject detail (sources, analyses, reports)	Auth
/app/subjects/:id/chat	Chat interface (update + query modes)	Auth
/app/admin/users	Manage Users	Group Admin
/app/admin/group-settings	Group Settings	Group Admin
/app/su/groups	Manage Groups	Superuser
/app/su/settings	Global Settings	Superuser
/app/su/subject-types	Subject Types + Template Management	Superuser
/app/logout	Logout	Auth

24. Subject Detail Page (Main Feature)

- **Action buttons**: Discover (Signal Agent) → Collect All → Analyze (Fusion) → Chat
- **Sources table**: Enable/disable toggles, collection status, frequency, play button, expandable run history
- **Config indicators**: Green check (URL configured) vs “needs config” (empty)
- **Analyses section**: Expandable cards with summary, key findings (severity badges), signals (confidence badges)
- **Reports section**: Expandable cards with rendered Markdown (react-markdown)

B. Key Design Patterns

- **Playbook-based provisioning**: Subject creation auto-provisions sources from type-matched templates; templates are fully editable and clonable

- **Agent orchestration via CrewAI**: 5 agents (Signal, Fusion, Quill, Ad-hoc Update, Ad-hoc Query) with Claude Sonnet, run as background tasks via *asyncio.to_thread*
- **Collector registry pattern**: *@register_collector("tool_name")* decorator; new collectors added with one function + one decorator
- **File-based data storage**: Collected data as JSON files in *data/{group}/{subject}/*; keeps database lean, files inspectable
- **CSS variable theming**: 21-color palette with 10 shades each, injected as CSS variables, utility classes (bg-tc-*, text-tc-*)
- **setTimeout polling** (Cognosa pattern): Backend status drives frontend polling; each poll waits for previous to complete; no *setInterval* race conditions
- **Backward-compatible properties**: Model refactoring (enum → FK) uses *@property* on models so all serialization continues to work
- **Hierarchical RBAC**: user < subjectmanager < groupadmin < superuser; enforced at API level with *_require_** helpers
- **Idempotent seeding**: Startup seeds subject types, playbook templates, admin user, and settings only if tables are empty
- **Repository pattern**: *backend/db/tables/* classes encapsulate all DB operations per table
- **JWT authentication**: 7-day tokens via fastapi-users with custom username-or-email login + group active check
- **NullPool for tests**: Prevents asyncpg "another operation in progress" errors in pytest

---

11. Configuration

1. Environment — *.env*

Variable	Purpose
DATABASE_URL	PostgreSQL async connection string
DATABASE_SYNC_URL	PostgreSQL sync connection string
SECRET	JWT signing key
CORS_ORIGINS	Comma-separated allowed origins
DEFAULT_ADMIN_PASSWORD	Initial admin password (default: "admin")
LOG_LEVEL	structlog level (default: INFO)
SCHEDULER_CHECK_INTERVAL_SECONDS	How often scheduler checks for due sources (default: 60)
ANTHROPIC_API_KEY	Anthropic API key for CrewAI agents

2. Database Settings — *api_settings* table

Setting	Purpose
app_title	Application title in navbar
navbar_color	Theme color (21 Tailwind color names)
instance_label	Badge text (DEV, STAGING, PROD)
dashboard_title	Dashboard banner title
dashboard_top	Dashboard banner subtitle
sw_ver	Software version string
db_ver	Database schema version

3. Group Settings — *group_settings* table

Setting	Purpose
reddit_client_id	Reddit API client ID (for praw collector)
reddit_client_secret	Reddit API client secret
enable_markdown_reports	Save reports as .md files (“true”/“false”)

---

12. Code Statistics

Metric	Count
Python files	81
TypeScript/TSX files	26
Python lines of code	~8,200
TypeScript lines of code	~3,100
Total lines of code	~11,300
Database tables	13
API route modules	11
CrewAI agents	5
Data collectors	4
Playbook templates	55
Test modules	12
Test cases	89
Alembic migrations	6

---

13. Build History

Sprint	Date	What was built
1	2026-04-01	Working Shell: FastAPI + React + auth + DB
2	2026-04-01	Subject management + playbook templates (55)
3	2026-04-01	Collection engine (crawl4ai, feedparser, httpx)
4	2026-04-01	CrewAI Signal Discovery Agent
5	2026-04-01	APScheduler + Dashboard
6	2026-04-01	Fusion analysis + Quill reports + Group Settings
7	2026-04-02	Ad-hoc chat (update + query modes)
8	2026-04-02	Dynamic subject types + editable templates
9	2026-04-02	Group + user management
+	2026-04-02	Settings CRUD, navbar theming, Reddit collector
+	2026-04-03	Landing page, health checks, dashboard banner

---

14. Local Development

```
# Prerequisites: PostgreSQL 17, Python 3.12, Node 18+

# Database setup
createdb cim_db
createdb cim_db_test

# Backend
cd compintelmon_code
pip install -r backend/requirements.txt
python3 -m playwright install chromium    # for crawl4ai
alembic upgrade head                      # apply migrations

# Frontend
cd frontend
npm install
npm run build                             # builds to ../backend/static/

# Run
python3 -m uvicorn backend.main:app --reload --port 8000

# Browse to http://localhost:8000/app/
# Login: admin / admin

# Run tests
python3 -m pytest backend/tests/ -v
```