Project Structure — Cognosa (Multi-Tenant RAG Platform)

1. Overview

Cognosa is a production-ready, multi-tenant Retrieval-Augmented Generation (RAG) platform. It enables organizations (tenants) to vectorize their proprietary documents, store them in tenant-isolated vector database collections, and give their users access to multiple open-source and proprietary LLMs for question answering against that data.

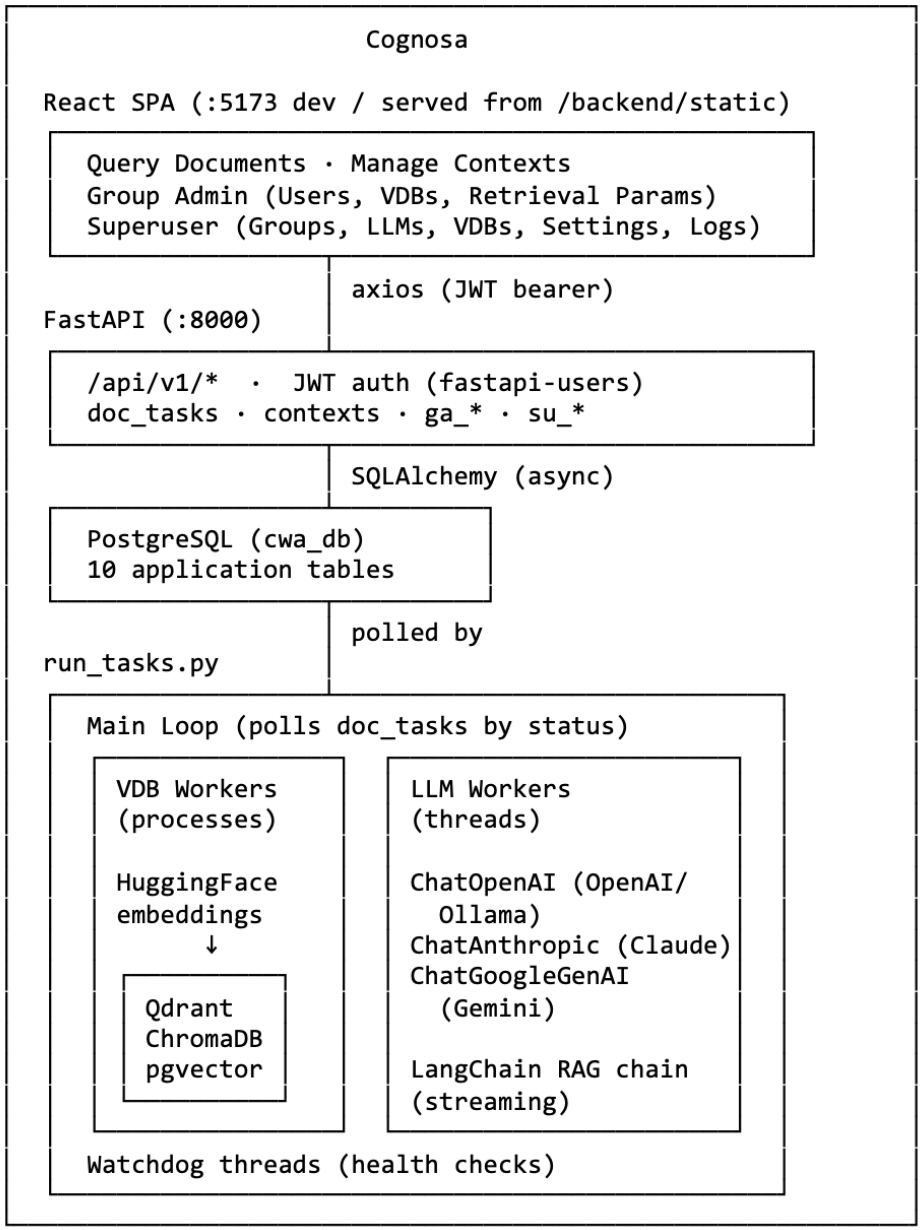
The platform has three runtime components:

1. **FastAPI web server** (*webapp.py*) — serves the React SPA and all REST API endpoints
2. **Background task processor** (*run_tasks.py*) — multiprocessing VDB workers + threaded LLM workers that execute the RAG pipeline
3. **React + TypeScript frontend** — Bootstrap-based UI with real-time query polling and streaming LLM response display

Version: 0.19a (2026-01-25) — see CHANGELOG.md for full history (development began August 2025).

Database: PostgreSQL (*cwa_db*), with optional Qdrant, ChromaDB, or pgvector for vector storage.

2. System Architecture



3. Tech Stack

A. Backend

Layer	Technology	Version
Language	Python	3.12
Web framework	FastAPI	0.116.1
ASGI server	Uvicorn	0.35.0
Auth	fastapi-users (JWT)	14.0.1
ORM	SQLAlchemy (async + sync)	2.0.43
Async DB driver	asyncpg	0.30.0
Sync DB driver	psycopg2-binary	2.9.10
Migrations	Alembic	1.17.1
LLM framework	LangChain	1.0.7
LLM: OpenAI/Ollama	langchain-openai	1.0.3
LLM: Anthropic	langchain-anthropic	1.1.0
LLM: Google	langchain-google-genai	3.1.0
VDB: Qdrant	qdrant-client + langchain-qdrant	1.15.1 / 1.1.0
VDB: ChromaDB	chromadb + langchain-chroma	1.0.20 / 1.0.0
VDB: pgvector	langchain-community (PGVector)	0.4.1
Embeddings	sentence-transformers (HuggingFace)	5.1.2
Token counting	tiktoken	0.11.0
Doc loaders	pypdf, docx2txt, python-pptx, unstructured, openpyxl	—
Templates	Jinja2	3.1.6

B. Frontend

Layer	Technology	Version
Framework	React	19.1
Language	TypeScript	5.8
Build tool	Vite	7.1
UI library	React Bootstrap	2.10
Icons	Bootstrap Icons	1.13
State management	Zustand	5.0
HTTP client	Axios	1.12
Routing	react-router-dom	7.8
Markdown	react-markdown + rehype-highlight	—
Excel export	SheetJS (xlsx)	0.20

C. Key Design Patterns

- **State machine task processing**: *doc_tasks.status* drives the RAG pipeline through well-defined stages; *run_tasks.py* polls by status and dispatches to appropriate workers
- **Worker pool architecture**: VDB workers use multiprocessing (avoids GIL for CPU-bound embedding), LLM workers use threading (I/O-bound network calls)
- **Config snapshots**: *doc_tasks* stores *gvdbs_json* and *gllms_json* at query time, preserving the exact config even if VDB/LLM settings change later
- **Cascading retrieval parameters**: Global → group → collection → query, with role-based visibility control
- **Soft deletes**: All tenant tables use *deleted* flag; user soft-delete renames email/username to prevent uniqueness conflicts on re-creation
- **Repository pattern**: *cwa_lib/sql_tables/* classes encapsulate all DB operations per table
- **Watchdog health monitoring**: Background threads periodically check VDB and LLM availability, updating status in *api_processes* and the per-resource status fields
- **JWT authentication**: 7-day tokens via fastapi-users with custom username-or-email login
- **Dependency injection**: FastAPI *Depends()* for session management and auth guards
- **Audit logging**: All CRUD operations logged to *log_crud* with user, IP, method, URL, data, and result

4. Multi-Tenancy Architecture

D. Tenant Isolation

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

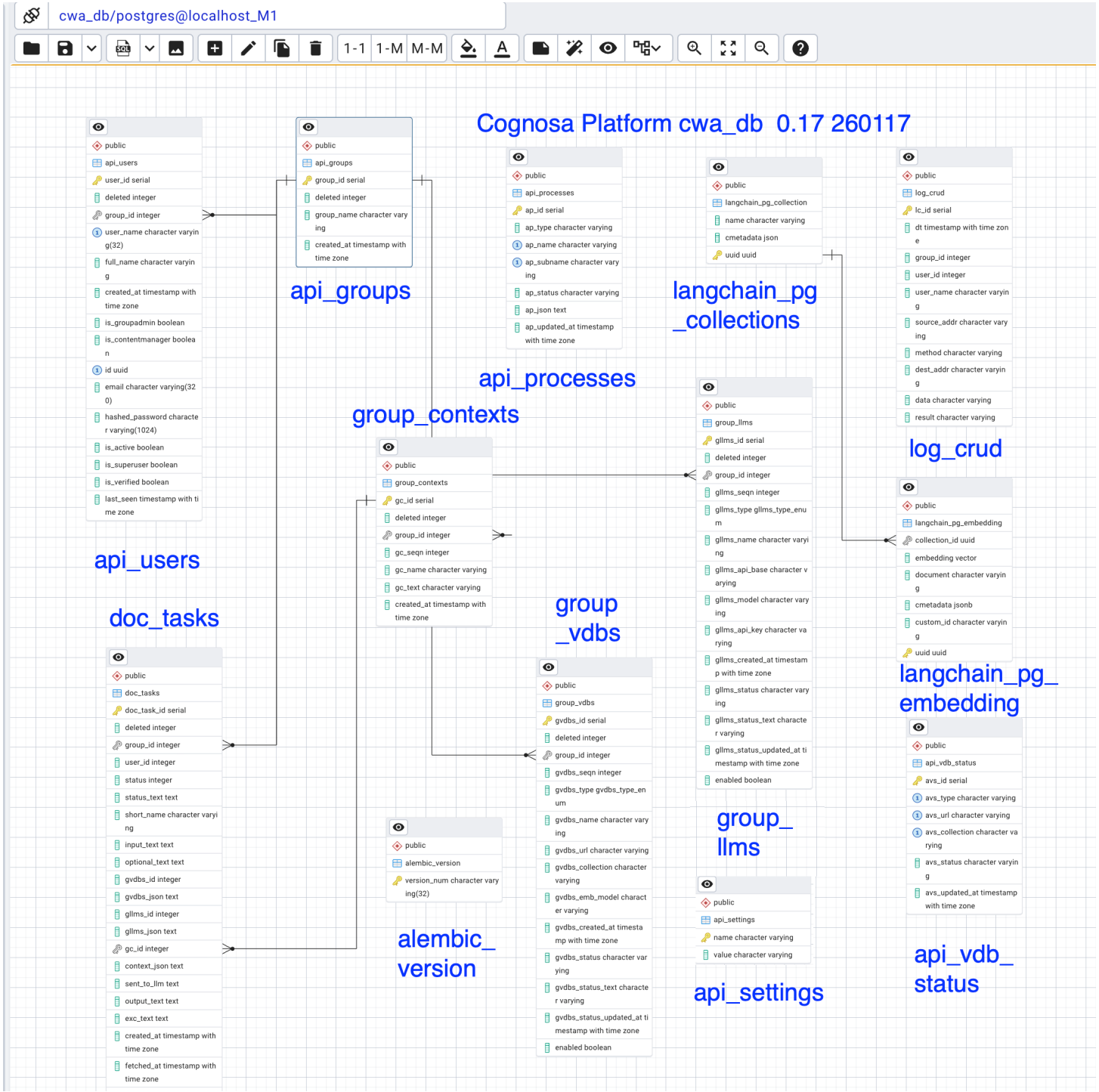
Level	Isolation Strategy
Database	All queries filter by *group_id*; FK constraints enforce relationships
Vector DB	Separate collections per group (user-defined *gvdbs_collection* name)
LLM configs	Each group has its own LLM entries with separate API keys
Prompt templates	Group-scoped *group_contexts* with customizable RAG prompts
Retrieval params	Cascading defaults: global → group → per-collection → per-query
Application	User→Group mapping via *api_users.group_id*; all routes check group

E. User Roles

Role	Flag	Capabilities
User	*is_active*	Create queries, view own query history, use predefined retrieval parameters
Content Manager	*is_contentmanager*	+ manage group prompt templates (CRUD), override retrieval parameters per query
Group Admin	*is_groupadmin*	+ manage group users, view all group queries, manage group VDB configs, edit group retrieval defaults
Superuser	*is_superuser*	Full system access: all groups, users, VDBs, LLMs, global settings, audit logs, server health

5. Database — *cwa_db*

PostgreSQL 17.x, 10 application tables + LangChain pgvector tables.


cwa_db/postgres@localhost_M1
1-1
1-M
M-M
Cognosa Platform cwa_db 0.17 260117
public
api_users
user_id serial
deleted integer
group_id integer
user_name character varying(32)
full_name character varying
created_at timestamp with time zone
is_groupadmin boolean
is_contentmanager boolean
id uuid
email character varying(320)
hashed_password character varying(1024)
is_active boolean
is_superuser boolean
is_verified boolean
last_seen timestamp with time zone
api_users
public
api_groups
group_id serial
deleted integer
group_name character varying
created_at timestamp with time zone
api_groups
public
api_processes
ap_id serial
ap_type character varying
ap_name character varying
ap_subname character varying
ap_status character varying
ap_json text
ap_updated_at timestamp with time zone
api_processes
public
langchain_pg_collection
name character varying
cmetadata json
uuid uuid
langchain_pg_collections
public
log_crud
lc_id serial
dt timestamp with time zone
group_id integer
user_id integer
user_name character varying
source_addr character varying
method character varying
dest_addr character varying
data character varying
result character varying
log_crud
group_contexts
public
group_contexts
gc_id serial
deleted integer
group_id integer
gc_seqn integer
gc_name character varying
gc_text character varying
created_at timestamp with time zone
public
group_llms
gllms_id serial
deleted integer
group_id integer
gllms_seqn integer
gllms_type gllms_type_enum
gllms_name character varying
gllms_api_base character varying
gllms_model character varying
gllms_api_key character varying
gllms_created_at timestamp with time zone
gllms_status character varying
gllms_status_text character varying
gllms_status_updated_at timestamp with time zone
enabled boolean
group_llms
public
langchain_pg_embedding
collection_id uuid
embedding vector
document character varying
cmetadata jsonb
custom_id character varying
uuid uuid
langchain_pg_embedding
doc_tasks
public
doc_tasks
doc_task_id serial
deleted integer
group_id integer
user_id integer
status integer
status_text text
short_name character varying
input_text text
optional_text text
gvdbs_id integer
gvdbs_json text
gllms_id integer
gllms_json text
gc_id integer
context_json text
sent_to_llm text
output_text text
exc_text text
created_at timestamp with time zone
fetched_at timestamp with time zone
group_vdbs
public
group_vdbs
gvdbs_id serial
deleted integer
group_id integer
gvdbs_seqn integer
gvdbs_type gvdbs_type_enum
gvdbs_name character varying
gvdbs_url character varying
gvdbs_collection character varying
gvdbs_emb_model character varying
gvdbs_created_at timestamp with time zone
gvdbs_status character varying
gvdbs_status_text character varying
gvdbs_status_updated_at timestamp with time zone
enabled boolean
public
alembic_version
version_num character varying(32)
alembic_version
public
api_settings
name character varying
value character varying
api_settings
public
api_vdb_status
avs_id serial
avs_type character varying
avs_url character varying
avs_collection character varying
avs_status character varying
avs_updated_at timestamp with time zone
api_vdb_status

1. Core Tables

Table	Purpose
api_groups	Tenant groups with default retrieval parameters (JSON)
api_users	Users (extends fastapi-users UUID model): email, username, group_id, role flags, last_seen. Soft delete renames email/username to *deleted_{id}__<original>*.
doc_tasks	RAG query tasks: input question, optional instruction, VDB/LLM config snapshots (*gvdbs_json*, *gllms_json*), retrieved context (JSON), sent prompt, LLM output (primary + follow-up), timing stats, token counts, status state machine
group_vdbs	Per-group vector DB configurations: type (chroma/qdrant/pgvector), URL, collection name, embedding model, retrieval params, health status, enable/disable
group_llms	Per-group LLM configurations: type (6 variants), API base URL, model name, API key, health status, enable/disable
group_contexts	Per-group prompt templates (must contain *{context}* and *{question}* placeholders)

2. System Tables

Table	Purpose
api_settings	Key-value global settings: *app_version*, *db_version*, *webapp_main_color* (22 Tailwind options), *index_page* (HTML), *gvdbs_def_retr_params* (JSON)
api_processes	Process health monitoring: type, name, sub-process, status, JSON metadata, last update
log_crud	CRUD audit log: timestamp, user, IP, HTTP method, URL, request data, result

3. pgvector Tables (auto-created by LangChain)

Table	Purpose
langchain_pg_collection	Vector collection metadata
langchain_pg_embedding	Vector embeddings with JSONB metadata

6. RAG Pipeline — Task State Machine

Queries progress through a status state machine (positive = progress, negative = error):

```
QD_INIT (0)            User submits query → doc_task created
    ↓
QD_INIT_FETCHED (1)    run_tasks validates task, resolves VDB/LLM configs
    ↓
QD_VDB_PENDING (2)     Dispatched to VDB worker queue
    ↓  [VDB Worker - separate process]
    ↓  Loads embedding model → queries vector DB → retrieves top-k documents
    ↓
QD_VDB_FETCHED (3)     Context documents saved to doc_tasks.context_json
    ↓
QD_LLM_PENDING (4)     Dispatched to LLM worker
    ↓  [LLM Worker - thread]
    ↓  Formats prompt (context + question via group_contexts template)
    ↓  Streams response via LangChain RAG chain
    ↓
QD_LLM_WRITING (5)     Partial answer streaming (updated per chunk)
    ↓
QD_LLM_FETCHED (6)     Complete — answer in doc_tasks.output_text

Error states:  QD_INIT_ERROR (-1)  |  QD_VDB_ERROR (-3)  |  QD_LLM_ERROR (-6)
```

The frontend polls *GET /api/v1/doc_tasks/{id}* to display streaming progress.

F. Follow-up Questions

Users can ask a second question on the same retrieved context. The follow-up answer is stored in *doc_tasks.output_text_2* with *question_number = 2*.

---

7. LLM Support

G. Supported Types

Type	LangChain Class	Notes
dummy	(testing)	Returns canned response, no external calls
ollama_local	ChatOpenAI	Local Ollama instance (custom *base_url*)
ollama_remote	ChatOpenAI	Remote Ollama instance
chatgpt	ChatOpenAI	OpenAI API (empty *base_url* = default)
claude	ChatAnthropic	Anthropic API
gemini	ChatGoogleGenerativeAI	Google API

H. LLM Configuration

- **Per-group**: Each tenant configures their own LLMs with display name, model, API key, and display order
- **API keys**: Stored in *group_llms.gllms_api_key*
- **Health checks**: Local LLMs checked every 60s; public APIs (ChatGPT, Gemini, Claude) every 5 minutes
- **Streaming**: All LLMs support token streaming; temperature fixed at 0.0; max 10,000 tokens
- **Prompt capture**: Full prompt saved to *doc_tasks.sent_to_llm* for debugging/auditing
- **Token counting**: Estimated via tiktoken (gpt4o model as baseline)

-

I. RAG Chain (LangChain)

```
rag_chain = (
    {"context": lambda x: full_context, "question": RunnablePassthrough()}
    | PromptTemplate.from_template(template)   # from group_contexts
    | RunnableLambda(capture_prompt)            # saves to sent_to_llm
    | llm                                       # ChatOpenAI / ChatAnthropic / etc.
    | StrOutputParser()
)
for chunk in rag_chain.stream(query_text):
    # update doc_tasks.output_text incrementally
```

8. Vector Database Support

1. Supported Backends

Type	Client	Notes
qdrant	QdrantVectorStore (LangChain) + QdrantClient	COSINE distance, collection auto-creation
chroma	Chroma (LangChain) + HttpClient	HTTP connection to ChromaDB server
pgvector	PGVector (LangChain community)	Uses existing PostgreSQL instance

2. Embedding Models

Default: *sentence-transformers/all-MiniLM-L6-v2* (HuggingFace). Configurable per *group_vdbs.gvdbs_emb_model*. Models are preloaded in VDB worker processes for performance.

3. Retrieval Parameters

Configurable at four levels (cascading): global default → group default → per-collection → per-query override.

```
{
  "search_type": "similarity | mmr | similarity_score_threshold",
  "search_kwargs__similarity": { "k": 10 },
  "search_kwargs__mmr": { "k": 10, "fetch_k": 20, "lambda_mult": 0.5 },
  "search_kwargs__similarity_score_threshold": { "k": 10, "score_threshold": 0.5 }
}
```

Regular users use the collection defaults silently. Content managers, group admins, and superusers can override per query.

9. Document Ingestion

Standalone multiprocessing scripts (not part of the web app runtime):

```
python document_loader_qdrant.py      # → Qdrant collection
python document_loader_pgvector.py    # → pgvector tables
python tools/document_loader_chroma.py  # → ChromaDB collection
```

Note: Multiple off=the-shelf ingestions patterns have been implemented specifically for various data formats and types, including

J. Supported Document Formats

Format	LangChain Loader
PDF	PyPDFLoader
Word (.docx)	Docx2txtLoader
PowerPoint (.pptx)	UnstructuredPowerPointLoader
HTML	UnstructuredHTMLLoader
Plain text	TextLoader
CSV	CSVLoader
Excel (.xlsx)	UnstructuredExcelLoader

K. Chunking

```
RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
```

L. Process

1. Scan folder for supported document types
2. Spawn N worker processes (up to CPU cores - 1)
3. Each worker: load document → split into chunks → embed with HuggingFace model → save to vector DB collection
4. File path stored in metadata for provenance tracking

10. API Routes

Base prefix: `/api/v1`

M. 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)

N. Document Queries (authenticated)

Method	Path	Description
POST	`/doc_tasks`	Create RAG query (question, optional instruction, VDB, LLM, context template)
GET	`/doc_tasks/{id}`	Get task status/result (polled for streaming)
DELETE	`/doc_tasks/{id}`	Delete own query
POST	`/doc_tasks/query_short`	Get user's query history (grouped: today/this week/before)
GET	`/doc_tasks/options`	Available VDBs, LLMs, and contexts for current user's group

O. Manage Contexts (authenticated)

Method	Path	Description
GET/POST/PUT/DELETE	`/manage_contexts[/{id}]`	CRUD for group prompt templates

P. Group Admin (`is_groupadmin`)

Prefix	Resource	Description
`/ga/users`	Group users	CRUD (cannot edit superusers or own superuser flag)
`/ga/doc_tasks`	Group queries	View all queries for own group
`/ga/vdbs`	Group VDBs	Edit enabled, name, display order, retrieval params
`/ga/settings`	Retrieval params	Edit group-level default retrieval parameters

Prefix	Resource	Description

Q. Superuser (*is_superuser*)

Prefix	Resource	Description
/su/groups	All groups	CRUD
/su/users	All users	CRUD across all groups
/su/llms	All LLMs	CRUD (type, model, API key, URL, enable/disable)
/su/vdbs	All VDBs	CRUD (type, URL, collection, embedding model, retrieval params)
/su/api_settings	Global settings	Edit app settings (theme, version, index page, default retrieval params)
/su/doc_tasks	All queries	View/delete queries across all groups
/su/log_crud	Audit log	View CRUD operation log
/su/server_status	Health dashboard	VDB/LLM/process status
/su/change_oneself	Self-management	Change own group_id and role flags

R. Public

Method	Path	Description
GET	*/public-settings*	Client name, app/db version (for login page)
GET	*/health*	Database connectivity check

11. Frontend Pages

S. URL Routes

Path	Page	Access
/	Index page (from *api_settings.index_page*)	Public
/login	Login form (shows client name, version)	Public
/app	Home page (VDB/LLM status, version info)	Auth
/app/query	Query Documents (main RAG interface)	Auth
/app/contexts	Manage Contexts	Auth
/app/ga/users	Group Admin: Users	Group Admin
/app/ga/vdbs	Group Admin: VDBs	Group Admin
/app/ga/settings	Group Admin: Retrieval Parameters	Group Admin
/app/ga/doc_tasks	Group Admin: Queries	Group Admin
/app/su/groups	Superuser: Groups	Superuser
/app/su/users	Superuser: Users	Superuser
/app/su/llms	Superuser: LLMs	Superuser
/app/su/vdbs	Superuser: VDBs	Superuser
/app/su/api_settings	Superuser: App Settings	Superuser
/app/su/doc_tasks	Superuser: All Queries	Superuser
/app/su/log_crud	Superuser: Audit Log	Superuser
/app/su/server_status	Superuser: Server Status	Superuser
/app/su/change_oneself	Superuser: Change Group/Roles	Superuser

T. Query Documents Page (Main Feature)

- **Left panel**: Query history grouped by Today / This Week / Before, with delete buttons
- **Right panel**: Query form (select VDB collection, LLM, context template, enter question + optional instruction)
- **Retrieval Parameters**: Configurable search type (similarity/MMR/SST) with per-type kwargs; visible only to content managers and above
- **Results**: Streamed markdown with syntax highlighting; shows VDB/LLM timing and token counts
- **Found Documents**: Modal showing retrieved context chunks (superuser only)
- **Follow-up**: Second question on same context, answer stored separately
- **Clone**: Duplicate a previous query for modification
- **No Document Search**: Option to query LLM directly without vector retrieval

U. Generic CRUD Tables

All management pages use a shared generic table component with: - Sortable columns, pagination (5/10/20 per page) - Create/Update modals with field descriptions and validation - View modal (vertical layout for wide rows) - Export to Excel/JSON - Reload button, dimmed cells during loading

---

12. Deployment

V. Docker Compose (Production)

File: *release/ec2_ubuntu_24_04/cognosa/docker-compose.yml*

Service	Image	Port	Purpose
db	postgres:17	5432	PostgreSQL database
qdrant	qdrant/qdrant:v1.15.5	6333	Qdrant vector database
app	(built from webapp.Dockerfile)	8000 (internal)	FastAPI web server

Service	Image	Port	Purpose
rt	(built from run_tasks.Dockerfile)	—	Background task processor
nginx	nginx:stable	80, 443	Reverse proxy + HTTPS
certbot	certbot/certbot	—	Let's Encrypt SSL certificates

All services have log rotation (10MB max, 3 files). Nginx proxies to *app:8000*. SSL certificates shared via Docker volumes.

W. Ollama Service

Separate systemd deployment: *release/ec2_ubuntu_24_04/cognosa_llm/ollama.service*

1. Local Development

```
# Terminal 1: Ollama (optional, for local LLM)
ollama run gemma3

# Terminal 2: Background task processor
cd backend && python3 run_tasks.py

# Terminal 3: FastAPI web server
cd backend && uvicorn webapp:app

# Terminal 4: Vector DB (Qdrant or ChromaDB)
qdrant                                  # Qdrant
chroma run --port 8010 --path ./chroma_db  # ChromaDB

# Browse to http://127.0.0.1:8000
```

X. Configuration

1. Environment — *backend/.env*

Variable	Purpose
DATABASE_URL	PostgreSQL connection (auto-converted to async/sync URLs)
SECRET	JWT signing key
CORS_ORIGINS	Comma-separated allowed origins
URL_QDRANT_LOCAL	Qdrant server address
URL_PG_LOCAL	pgvector PostgreSQL address
URL_OLLAMA_LOCAL	Ollama API endpoint
URL_CHROMA_LOCAL	ChromaDB server address
RT_VDB_PROCESS_NUM	Number of VDB worker processes (default: 1)
RT_VDB_EMB_MODELS_PRELOAD	Comma-separated embedding models to preload
LOG_SQLALCHEMY_RT	SQLAlchemy logging level for run_tasks

2. Database Settings — *api_settings* table

Setting	Purpose
app_version	Application version string
db_version	Database schema version
webapp_main_color	Theme color (22 Tailwind options)
index_page	HTML content for public index page
gvdbs_def_retr_params	Global default retrieval parameters (JSON)

3. Seed Data — *backend/.init_sql_data/*

JSON files loaded by *init_sql_db.py* for initial database population: groups, users, VDB configs, LLM configs, contexts, and settings.

13. Directory Layout

```
cognosa_web_app/
├── backend/
│   ├── webapp.py                    # FastAPI entry point (uvicorn webapp:app)
│   ├── run_tasks.py                 # Background task processor entry point
│   ├── init_sql_db.py               # Database initialization (loads .init_sql_data/)
│   ├── create_user.py               # Standalone user creation utility
│   ├── document_loader_qdrant.py    # Multiprocessing doc ingestion → Qdrant
│   ├── document_loader_pgvector.py  # Multiprocessing doc ingestion → pgvector
│   ├── .env                         # Environment config (DB, secrets, URLs)
│   ├── alembic.ini                  # Alembic migration config
│   │
│   ├── common/                      # Shared utilities & models
│   │   ├── __init__.py              #   Config loading, constants (API_URL_PREFIX,
│   │   │                            #     DATABASE_URL, CORS_ORIGINS, etc.)
│   │   ├── async_log.py             #   AsyncLogger utility
│   │   ├── helpers.py               #   Helper functions (split2list, etc.)
│   │   ├── parsed_url.py            #   URL parsing for named endpoints
│   │   ├── sql_db_async.py          #   Async SQLAlchemy session factory
│   │   ├── sql_db_sync.py           #   Sync SQLAlchemy session factory
│   │   ├── sql_tools.py             #   SQL query utilities
│   │   │
│   │   ├── enums/                   #   Enum constants
│   │   │   ├── api_settings_names.py #     Setting name constants
│   │   │   ├── doc_task_status.py   #     TaskStatus state machine (0→6, errors <0)
│   │   │   ├── gllms_types.py       #     LLM types: dummy, ollama_local/remote,
│   │   │   │                        #       chatgpt, gemini, claude
│   │   │   └── gvdbs_types.py       #     VDB types: chroma, qdrant, pgvector
│   │   │
│   │   ├── features/
│   │   │   └── gvdbs_retr_params.py #   Retrieval parameter defaults & merging
│   │   │
│   │   ├── sql_models/              #   SQLAlchemy ORM models
│   │   │   ├── api_groups.py        #     Tenant groups
│   │   │   ├── api_users.py         #     Users (extends SQLAlchemyBaseUserTableUUID)
```

```
│   │   │   ├── api_processes.py       #     Process health monitoring
│   │   │   ├── api_settings.py        #     Key-value app settings
│   │   │   ├── doc_tasks.py           #     RAG query tasks
│   │   │   ├── group_contexts.py      #     Prompt templates per group
│   │   │   ├── group_llms.py          #     LLM configs per group
│   │   │   ├── group_vdbs.py          #     VDB configs per group
│   │   │   └── log_crud.py            #     CRUD audit log
│   │   │
│   │   └── watchdogs/                 #   Health monitoring threads
│   │       ├── watchdog_thread.py     #     Base watchdog thread class
│   │       ├── api_processes_table.py #     Process table updater
│   │       ├── group_llms.py          #     LLM availability checker
│   │       └── group_vdbs.py          #     VDB availability checker
│   │
│   ├── cwa_lib/                       # Core web application library
│   │   ├── app.py                     #   FastAPI app factory
│   │   ├── users.py                   #   fastapi-users config (JWT strategy,
│   │   │                              #     custom username auth, current_active_user)
│   │   │
│   │   ├── middleware/
│   │   │   └── last_seen.py           #   Update user.last_seen on each request
│   │   │
│   │   ├── routers/                   #   API route definitions
│   │   │   ├── __init__.py            #     Route aggregation (api_router)
│   │   │   ├── doc_tasks.py           #     POST/GET/DELETE doc_tasks, GET options
│   │   │   ├── users.py              #     GET /users/me
│   │   │   ├── manage_contexts.py     #     CRUD for group prompt templates
│   │   │   ├── misc.py                #     Public settings, health check
│   │   │   ├── webapp_options.py      #     Frontend config options
│   │   │   ├── ga_manage_users.py     #     Group admin: user management
│   │   │   ├── ga_manage_doc_tasks.py #     Group admin: view group queries
│   │   │   ├── ga_manage_vdbs.py      #     Group admin: VDB config management
│   │   │   ├── ga_settings.py         #     Group admin: retrieval parameters
│   │   │   ├── su_manage_groups.py    #     Superuser: group CRUD
│   │   │   ├── su_manage_users.py     #     Superuser: all-user management
│   │   │   ├── su_manage_llms.py      #     Superuser: LLM config CRUD
│   │   │   ├── su_manage_vdbs.py      #     Superuser: VDB config CRUD
│   │   │   ├── su_manage_api_settings.py  # Superuser: global settings
```

```
│   │   │   ├── su_manage_doc_tasks.py #     Superuser: view/delete all queries
│   │   │   ├── su_manage_log_crud.py  #     Superuser: audit log viewer
│   │   │   ├── su_server_status.py    #     Superuser: system health dashboard
│   │   │   └── su_change_oneself.py   #     Superuser: change own group/roles
│   │   │
│   │   ├── pages/                     #   Business logic handlers (called by routers)
│   │   │   ├── query_documents.py     #     RAG query creation & retrieval
│   │   │   ├── manage_contexts.py     #     Context CRUD logic
│   │   │   ├── server_status.py       #     System status aggregation
│   │   │   ├── ga_manage_*.py         #     Group admin page handlers
│   │   │   └── su_manage_*.py         #     Superuser page handlers
│   │   │
│   │   ├── pydantic_schemas/          #   Request/response models
│   │   │   ├── user.py                #     UserRead, UserCreate
│   │   │   ├── doc_tasks.py           #     Task create/read schemas
│   │   │   ├── ga_*.py                #     Group admin schemas
│   │   │   └── su_*.py                #     Superuser schemas
│   │   │
│   │   ├── sql_tables/                #   Repository-pattern DB access
│   │   │   ├── api_groups.py
│   │   │   ├── api_settings.py
│   │   │   ├── api_users.py
│   │   │   ├── doc_tasks.py
│   │   │   └── log_crud.py
│   │   │
│   │   └── validators/                #   Input validation
│   │       ├── api_settings.py
│   │       ├── messages.py
│   │       ├── strings.py
│   │       └── user_name.py           #   3-32 chars, a-z0-9_- only
│   │
│   ├── tasks_lib/                     # Background task processing
│   │   ├── cmd_line_opts.py           #   CLI argument parsing (-s <instance>)
│   │   ├── main_iteration.py          #   Main polling loop (processes doc_tasks)
│   │   ├── vdb_llm_status_worker.py   #   Periodic VDB/LLM health checks
│   │   │
│   │   ├── entities/                  #   Inter-process message types
│   │   │   ├── task_queue_msg.py      #     VDB task queue message
```

```
│   │   │   └── llm_worker_msg.py      #     LLM worker message
│   │   │
│   │   ├── vdb_lib/                   #   Vector database integrations
│   │   │   ├── vdb_ops.py             #     VDB orchestration (dispatch by type)
│   │   │   ├── chromadb_ops.py        #     ChromaDB: connect, retrieve
│   │   │   ├── qdrant_ops.py          #     Qdrant: connect, retrieve, create collection
│   │   │   ├── pgvector_ops.py        #     pgvector: connect, retrieve
│   │   │   ├── emb_models.py          #     Embedding model manager (preload, cache)
│   │   │   ├── found_documents.py     #     Document result formatting
│   │   │   └── workers.py            #     VDB worker processes (multiprocessing)
│   │   │
│   │   ├── llm_lib/                   #   LLM integrations
│   │   │   ├── llm_ops.py             #     LLM orchestration (dispatch by type)
│   │   │   ├── llm_type_openai.py     #     ChatOpenAI (OpenAI + Ollama)
│   │   │   ├── llm_type_claude.py     #     ChatAnthropic
│   │   │   ├── llm_type_gemini.py     #     ChatGoogleGenerativeAI
│   │   │   ├── llm_type_dummy.py      #     Testing dummy LLM
│   │   │   ├── tiktoken_count.py      #     Token estimation (gpt4o model)
│   │   │   └── workers.py            #     LLM worker threads
│   │   │
│   │   └── qd_lib/                    #   Query Document processing stages
│   │       ├── qd_init.py             #     Task validation (QD_INIT → QD_INIT_FETCHED)
│   │       └── qd_vdb_fetched.py      #     Post-VDB processing (format context)
│   │
│   ├── .init_sql_data/                # Seed data for init_sql_db.py
│   │   ├── api_settings.json
│   │   ├── api_groups.json
│   │   ├── api_users.json
│   │   ├── group_vdbs.json
│   │   ├── group_llms.json
│   │   ├── group_contexts.json
│   │   └── doc_tasks.json
│   │
│   ├── alembic/                       # Database migrations
│   │   ├── env.py
│   │   └── versions/                  #   Incremental migration scripts
│   │
│   ├── static/                        # Compiled frontend assets (served by FastAPI)
```

```
│   ├── templates/                    # Jinja2 templates (index, login pages)
│   └── tools/                        # Utility scripts
│       ├── document_loader_chroma.py #   ChromaDB document ingestion
│       └── check_chroma_server.py    #   ChromaDB health check
│
├── frontend/                         # React + TypeScript SPA
│   ├── index.html
│   ├── package.json
│   ├── tsconfig.json
│   ├── vite.config.ts
│   │
│   └── src/
│       ├── main.tsx                  #   React entry point
│       ├── main.css                  #   Global styles
│       │
│       ├── api/                      #   Axios API client wrappers
│       ├── components/               #   Reusable UI components
│       ├── hooks/                    #   Custom React hooks
│       ├── stores/                   #   Zustand state stores
│       ├── tables/                   #   Generic CRUD table components
│       │
│       └── pages/
│           ├── App.tsx               #   Root component with routing
│           ├── HomePage/             #   Landing: VDB/LLM status, version info
│           ├── QueryDocuments/       #   RAG query interface (main feature)
│           ├── ManageContexts/       #   Prompt template CRUD
│           ├── GaManageUsers/        #   Group admin: user management
│           ├── GaManageVDBs/         #   Group admin: VDB config
│           ├── SuManageGroups/       #   Superuser: group management
│           ├── SuManageLLMs/         #   Superuser: LLM config
│           ├── SuManageVDBs/         #   Superuser: VDB config
│           └── ...                   #   Additional su_* pages
│
├── release/                          # Docker deployment configs
│   └── ec2_ubuntu_24_04/
│       ├── cognosa/                  #   Main app deployment
│       │   ├── docker-compose.yml    #     5 services: db, qdrant, app, rt, nginx
│       │   ├── docker-compose_no_nginx.yml
```

```
│       │   ├── webapp.Dockerfile
│       │   ├── run_tasks.Dockerfile
│       │   ├── nginx_default.conf
│       │   ├── install.sh
│       │   ├── update_ssl_cert.sh     #     Let's Encrypt renewal
│       │   └── env_*.env-default      #     Template env files
│       │
│       └── cognosa_llm/               #   Ollama LLM service
│           ├── cognosa_llm.sh
│           └── ollama.service         #     systemd unit file
│
├── docs/                              # Project documentation
├── snapshots/                         # Backup snapshots
├── requirements.txt                   # Python dependencies (33 packages)
├── README.md                          # Setup instructions
└── CHANGELOG.md                       # Version history (0.1 → 0.19a)
```

Y. Version History Highlights

Version	Date	Key Changes
0.1	2025-08-24	Initial prototype
0.2	2025-08-31	React frontend, user auth, login page
0.3	2025-09-07	PostgreSQL tables, multiprocessing VDB workers, streaming LLM
0.5	2025-09-21	API routes */api/v1*, query history, optional instruction text
0.6	2025-09-28	Multi-VDB support (chroma/qdrant/pgvector), document loaders
0.7	2025-10-04	Multi-LLM support, *group_llms* table, username auth
0.8	2025-10-12	Soft deletes, generic CRUD tables, audit logging
0.10c	2025-10-26	Claude LLM support, theming, group admin users page

Version	Date	Key Changes
0.13h	2025-11-16	Follow-up questions, no-document-search mode, Excel/JSON export
0.14g	2025-11-22	GitLab push, secrets cleanup, index page, */app* route
0.15f	2025-11-30	ChatGPT + Gemini LLM types, VDB/LLM status indicators
0.16d	2025-12-07	Alembic migrations, last_seen tracking, enabled flags
0.17b	2025-12-14	Docker log rotation, SSL cert renewal, auth fixes
0.18a	2025-01-18	Retrieval parameter refactoring (SIM/MMR/SST)
0.19a	2025-01-25	Per-group and per-collection retrieval parameters, group admin VDB page
0.20a	2026-02-01	Per group, per collection filtered retrieval capability added to database, UI (query docs and group collection attributes), and back end functions.
0.21b	2026-02-07	Implemented query filters and auto-filtering for specified collections (lbrprods)