Reactome LNP Agent

Technical Summary

A LangGraph-based Multi-Agent System for Ionizable Lipid Design, Synthesis Planning, and Property Prediction

Version 3.0 — February 2026

Abstract. This document describes the architecture and implementation of the Reactome LNP Agent v3, a full-stack web application for AI-assisted ionizable lipid design. Version 3 introduces a dual-model architecture: Claude 3.5 Haiku handles fast infrastructure tasks (query rewriting, routing, reranking) while Claude Sonnet 4.5 powers the five domain expert workers and the Lead Reasoning Agent. This tiered model assignment reduces cost by ~80% on high-frequency nodes and improves latency by ~20-30% with negligible quality impact. The system retains the 6-agent supervisor/worker architecture with YAML-driven prompt configuration from v2. The system combines Retrieval-Augmented Generation (RAG) with a FAISS vector store (33 research papers, 454 chunks) and Amazon Bedrock foundation models. The frontend has been updated with model attribution in the workflow visualization.

Component	Technology	Details
Frontend	Angular 21 + Tailwind CSS v4	SPA with SSE streaming, conversation persistence
Backend	FastAPI (Python 3.12)	REST + SSE, 7 endpoints
Orchestration	LangGraph	10-node DAG, 5 parallel workers + supervisor
Agent Config	YAML definitions	6 agents + system config + router
Mol. Analysis	RDKit	QED, SA Score, LogP, TPSA, 2D SVG
Vector DB	FAISS (local)	454 chunks, 1024-dim embeddings
Embeddings	Titan Embed Text v2	Amazon Bedrock
LLM (Experts + Lead)	Claude Sonnet 4.5	Amazon Bedrock, us-west-2
LLM (Router/Rewrite/Rerank)	Claude 3.5 Haiku	Amazon Bedrock, us-west-2
External Tools	PubMed, PubChem, Web	Live search via NCBI E-utils, PUG REST, DDG
Data	PDFs + CSV + SMARTS	33 papers, 13 reactions, 217K blocks

Component	Technology	Details
Standalone Web	Dioxus 0.6 (WASM)	Trunk-built, Python-served, RDKit analysis
Desktop App	wry 0.47 + tao (Rust)	Native webview, embedded Angular build

Table 1: Technology stack overview.

1. System Architecture

The Reactome LNP Agent v2 follows a three-tier architecture: an Angular single-page application communicates with a FastAPI backend via REST and Server-Sent Events (SSE), which orchestrates a LangGraph workflow leveraging both a local FAISS vector store and remote Amazon Bedrock foundation models. Version 2 introduces a fundamental architectural change: the agent pipeline is now driven by YAML configuration files (src/agents/*.yaml) that define each agent's identity, system prompt, model parameters, and tool access. A Python loader module (src/agents/__init__.py) reads these configs at startup and injects global constraints from the system.yaml master configuration into each agent's prompt.

The choice of YAML-driven configuration enables rapid iteration on agent behavior without code changes—modifying a prompt, adjusting temperature, or adding a new agent requires only editing a YAML file. The system.yaml file serves as the master configuration, defining the agent roster, routing rules, and six global constraints that are automatically appended to every agent's system prompt.

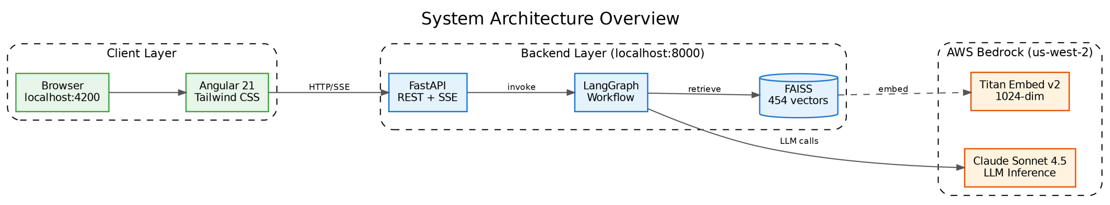

Figure 1. System architecture showing the three-tier design: Angular client, FastAPI backend with LangGraph and FAISS, and AWS Bedrock cloud services.

2. Agent System Architecture

Version 2 replaces the previous 3-expert pipeline with a 6-agent supervisor/worker system. Each agent is defined in a dedicated YAML file under src/agents/, specifying its role, system prompt, model parameters, output field, and available tools. The system.yaml master config defines global constraints, the agent roster with positions, and routing rules that map query types to expert subsets.

2.1 Agent Roster

Agent	Role	Position	Output Field
Router	Query classifier (synthesis / lookup / general)	Node 2	query_type
Reaction Expert	SMARTS template matching, feasibility assessment	Parallel worker	reaction_analysis
Lipid Design Expert	Retrosynthesis + SAR + design rule validation	Parallel worker	lipid_design_analysis
Generative AI Expert	De novo generation + RL optimization	Parallel worker	generative_analysis
Property Prediction Expert	ML model recommendations + uncertainty quantification	Parallel worker	prediction_analysis
Literature Scout	PubMed, PubChem, web search	Parallel worker	literature_context
Lead Agent	Supervisor — critically evaluates all expert outputs	Supervisor	final_answer

Table 2: Agent roster. Five parallel workers feed into the Lead Agent supervisor.

2.2 YAML Configuration

Each agent YAML file follows a consistent schema. The system.yaml master config defines global constraints that are automatically prepended to every agent's system prompt at load time:

```yaml
# system.yaml (excerpt)
system:
  name: "Reactome LNP Agent"
  version: "3.0"
  default_model: "us.anthropic.claude-sonnet-4-5-20250929-v1:0"
  fast_model: "us.anthropic.claude-3-5-haiku-20241022-v1:0"

  global_constraints:
    - "Never fabricate SMILES - only use validated structures"
    - "Always cite Mogam reaction template IDs (10001-10017)"
    - "Flag any compound with SA Score > 6 as difficult to synthesize"
    - "Use IUPAC nomenclature for chemical names"
    - "Express uncertainty explicitly"
    - "Distinguish computational predictions from experimental evidence"

  routes:
    synthesis:
      experts: [reaction_expert, lipid_design_expert,
               generative_ai_expert, property_prediction_expert,
               literature_scout]
    lookup:
      experts: [literature_scout]
    general:
      experts: [literature_scout]
```

2.3 Agent Merging Rationale

The v2 agent design consolidates related capabilities into fewer, more capable agents. The Lipid Design Expert merges three previously separate concerns (retrosynthesis planning, structure-activity relationships, and design rule validation) because a senior lipid chemist evaluates route, properties, and constraints simultaneously—they are inseparable aspects of design evaluation. Similarly, the Generative AI Expert merges molecular generation and RL optimization because generation and fine-tuning are two halves of the same pipeline. The Reaction Expert remains standalone because SMARTS pattern matching is a distinct technical skill requiring focused analysis.

2.4 Query Routing

The Router agent classifies each query into one of three categories, determining which experts are activated:

Category	Description	Experts Activated
synthesis	Design, build, optimize, or generate lipids	All 5 parallel workers
lookup	Specific fact about a known compound or reaction	Literature Scout only
general	Broad concepts, comparisons, recent research	Literature Scout + Web Search

Table 3: Query routing rules. Synthesis queries activate the full expert panel.

3. LangGraph Agent Pipeline

The core intelligence is implemented as a LangGraph StateGraph with 10 nodes. The pipeline begins with query rewriting (making questions self-contained using chat history), followed by routing classification, FAISS retrieval with LLM-based reranking, parallel expert execution, and final synthesis by the Lead Agent. For synthesis queries, all five expert workers execute in parallel, reducing latency compared to sequential execution. Lookup and general queries activate only the relevant subset of experts.

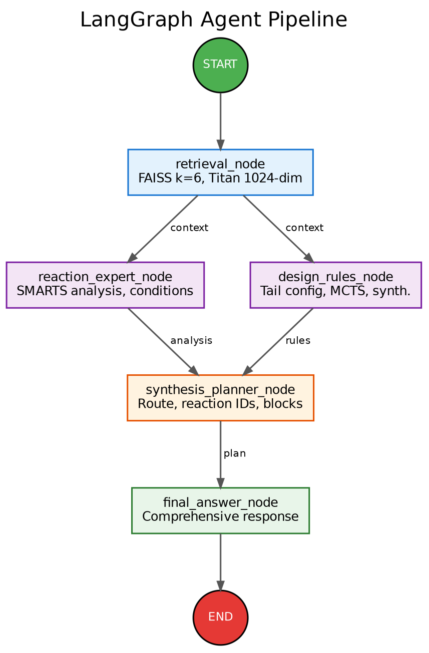

Figure 2. LangGraph pipeline. After retrieval, up to 5 expert workers execute in parallel before converging into the Lead Agent.

3.1 State Schema

The shared state carries data between all nodes. Each node reads from and writes to specific fields:

```
class ReactomeState(TypedDict):
    query: str # User's question
    chat_history: str # Previous conversation turns
    rewritten_query: str # Self-contained rewrite
    query_type: str # synthesis | lookup | general
    retrieved_context: str # Top-4 reranked FAISS chunks
    reaction_analysis: str # Reaction expert output
    lipid_design_analysis: str # Lipid design expert output
    generative_analysis: str # Generative AI expert output
    prediction_analysis: str # Property prediction expert output
    literature_context: str # PubMed + PubChem results
    web_context: str # Web search results
    final_answer: str # Lead agent's synthesized response
    error: str # Error capture
```

3.2 Node Specifications

Node	Input Fields	Output Field	Model
rewrite_query	query, chat_history	rewritten_query	Haiku 3.5
router	rewritten_query	query_type	Haiku 3.5

Node	Input Fields	Output Field	Model
retrieve + rerank	rewritten_query	retrieved_context	Haiku 3.5
reaction_expert	query, retrieved_context	reaction_analysis	Sonnet 4.5
lipid_design_expert	query, retrieved_context	lipid_design_analysis	Sonnet 4.5
generative_ai_expert	query, retrieved_context	generative_analysis	Sonnet 4.5
property_prediction_expert	query, retrieved_context	prediction_analysis	Sonnet 4.5
literature_search	rewritten_query	literature_context	API only
web_search	rewritten_query	web_context	API only
lead_agent	all analysis fields	final_answer	Sonnet 4.5

Table 4: Node specifications. Haiku 3.5 handles 3 fast nodes; Sonnet 4.5 handles 5 expert + lead.

3.3 Retrieval and Reranking

The retrieval node fetches the top-8 candidates from FAISS, then uses an LLM call to rerank them by relevance, selecting the top-4 chunks. This two-stage approach improves precision over raw vector similarity alone, as the LLM can assess semantic relevance that embedding distance may miss. The reranked chunks are concatenated with source type metadata and passed to all expert nodes.

3.4 Conditional Routing

After retrieval, a conditional edge dispatches to different expert subsets based on query_type. This avoids unnecessary LLM calls for simple lookups:

- **synthesis** → reaction_expert + lipid_design_expert + generative_ai_expert + property_prediction_expert + literature_search (5 parallel)
- **lookup** → literature_search only (1 node)
- **general** → web_search + literature_search (2 parallel)

4. Expert Agent Details

4.1 Reaction Compatibility Expert

Specializes in SMARTS template matching and reaction feasibility assessment. For each applicable template, it reports the reaction ID, functional group match, compatibility level (HIGH/MEDIUM/LOW), conditions, competing reactions, and known issues. It enforces the critical rule of never using reactions 10012 or 10017 (flagged as invalid).

4.2 Lipid Design Expert

Merges retrosynthesis planning, structure-activity relationships, and design rule validation into a single senior chemist perspective. Outputs include synthesis routes with reaction IDs, property profiles (pKa, LogP, MW, TPSA, SA Score), design rule compliance tables (PASS/FAIL per constraint), and comparisons to reference lipids (DLin-MC3-DMA, ALC-0315, SM-102).

4.3 Generative AI Expert

Covers de novo molecular generation (REINVENT, JT-VAE, MoLeR, diffusion models) and RL optimization (REINFORCE, PPO, DPO, MCTS). Provides a multi-objective reward function specification for ionizable lipids with hard constraints (valid SMILES, ionizable nitrogen, MW 500–1200, synthesizable via Mogam templates) and soft objectives (pKa, SA Score, LogP, liver targeting, novelty, diversity) with specific weights.

4.4 Property Prediction Expert

Recommends ML models for molecular property prediction based on data size and property type. Covers molecular representations (SMILES, fingerprints, graphs, 3D), model architectures (GNN, Transformers, RF/XGBoost, Gaussian Process), and transfer learning pipelines. Always requires uncertainty quantification for novel structures.

4.5 Literature Scout

Retrieves external evidence from three sources using live API calls:

Source	API	Output
PubMed	NCBI E-utilities (esearch + esummary)	Title, journal, year, PMID
PubChem	PUG REST (compound/name/property)	IUPAC name, SMILES, MW, formula
Web	DuckDuckGo HTML scraping	Title, snippet, URL

Table 5: External search tools available to the Literature Scout.

4.6 Lead Reasoning Agent

The supervisor agent receives all expert outputs and produces the final user-facing response. It applies critical evaluation rules: reaction expert overrides design expert on compatibility conflicts, design rule violations are treated as critical rejections, AI recommendations are presented as computational suggestions requiring validation, and confidence levels (HIGH/MEDIUM/LOW) are assigned based on evidence strength.

5. RAG System

The Retrieval-Augmented Generation system ingests domain-specific documents from five source types, processes them through a chunking pipeline, and stores the resulting vectors in a FAISS index. At query time, the user's question is embedded using the same Titan model and the top-8 most similar chunks are retrieved, then reranked by the LLM to select the top-4. Each chunk carries metadata (source_type) that helps the LLM understand provenance.

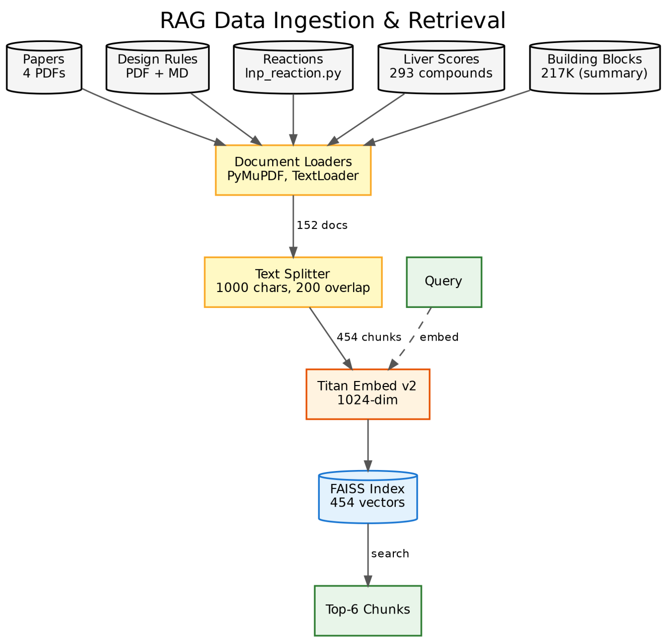

Figure 3. RAG data flow: five source types are loaded, chunked, embedded via Titan, and indexed in FAISS.

5.1 Data Sources

Source	**Type**	**Size**	**Content**
Research Papers	PDF	3 files	Lipid generation, SyntheMol-RL, MCTS approaches
Related Papers	PDF	30 files	LNP design, ML for lipids, diffusion models, transfection
LNP Design Rules	PDF + MD	2 files	MCTS tree structure, tail constraints, action space definitions
Reaction Templates	Python	1 file	13 Mogam SMARTS-based reaction definitions
Liver Scores	CSV	293 rows	SMILES with liver targeting scores
Building Blocks	CSV	217K rows	Head group building blocks (summary indexed)

Table 6: Data sources ingested into the RAG system.

5.2 Index Parameters

Parameter	**Value**
Text splitter	RecursiveCharacterTextSplitter

Parameter	Value
Chunk size / overlap	1,000 / 200 characters
Total documents $\rightarrow$ chunks	152 $\rightarrow$ 454
Embedding model	amazon.titan-embed-text-v2:0 (1,024-dim)
Index type	FAISS Flat L2
Initial retrieval k	8 (reranked to top 4)
Persisted index size	1.8 MB (index.faiss) + 477 KB (index.pkl)

Table 7: FAISS index configuration. Retrieval now uses 8+rerank strategy.

6. AWS Bedrock Integration

Amazon Bedrock serves as the inference backbone. Version 3 introduces a dual-model architecture: Claude 3.5 Haiku for fast infrastructure tasks (router, query rewrite, reranking) and Claude Sonnet 4.5 for expert reasoning and the Lead Agent. Both are accessed through langchain-aws wrappers. Two LLM instances are configured: a standard instance (Sonnet 4.5, max_tokens=4096) for the Lead Agent and expert workers, and a fast instance (Haiku 3.5, max_tokens=1024) for the Router, Rewrite, and Reranking nodes. This tiered approach reduces cost by ~80% on high-frequency nodes with negligible quality impact.

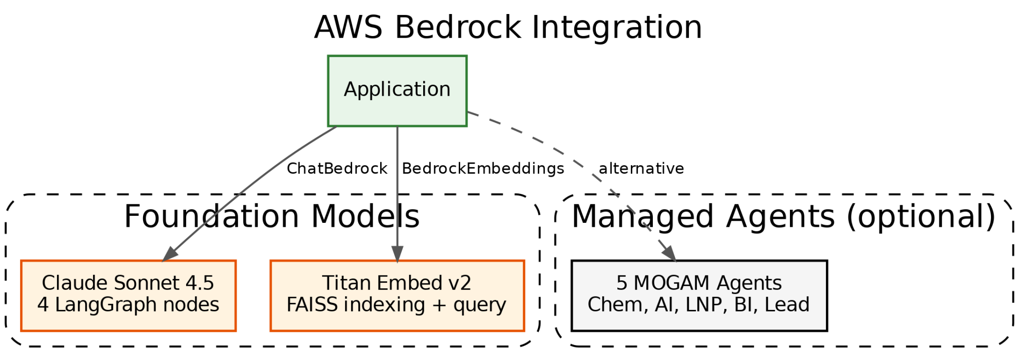

Figure 4. AWS Bedrock integration showing foundation models and optional managed agents.

6.1 Foundation Models

Model	**Model ID**	**Usage**
Claude Sonnet 4.5	us.anthropic.claude-sonnet-4-5-20250929-v1:0	Expert workers + Lead Agent (reasoning)
Claude 3.5 Haiku	us.anthropic.claude-3-5-haiku-20241022-v1:0	Router, query rewrite, reranking (fast)
Titan Embed Text v2	amazon.titan-embed-text-v2:0	Document + query embeddings (1024-dim)

Table 8: Bedrock foundation models. Dual-model architecture introduced in v3.

6.2 Managed Bedrock Agents

Five domain-specific agents are pre-configured in Bedrock for the multi-agent meeting workflow:

Agent	**ID**	**Role**
MOGAM-Chem-Agent	0IPX1MMI2D	Chemical structure and synthesis expert
MOGAM-AI-Agent	KQ9FJVLHQE	AI/ML methodology and model design
MOGAM-LNP-Agent	Q8GN0FK7NV	LNP formulation and delivery optimization
MOGAM-BI-Agent	XDPBOQN8YT	Bioinformatics and data analysis
MOGAM-Lead-Agent	RHUTNOTET1	Team lead, synthesis of expert inputs

Table 9: Managed Bedrock Agents for the MOGAM research team.

7. Backend API

The FastAPI backend exposes seven endpoints. The /api/chat endpoint streams status messages as each pipeline node begins execution, including the new agent names (lipid_design_expert, generative_ai_expert, property_prediction_expert). The details SSE event now includes all six expert output fields.

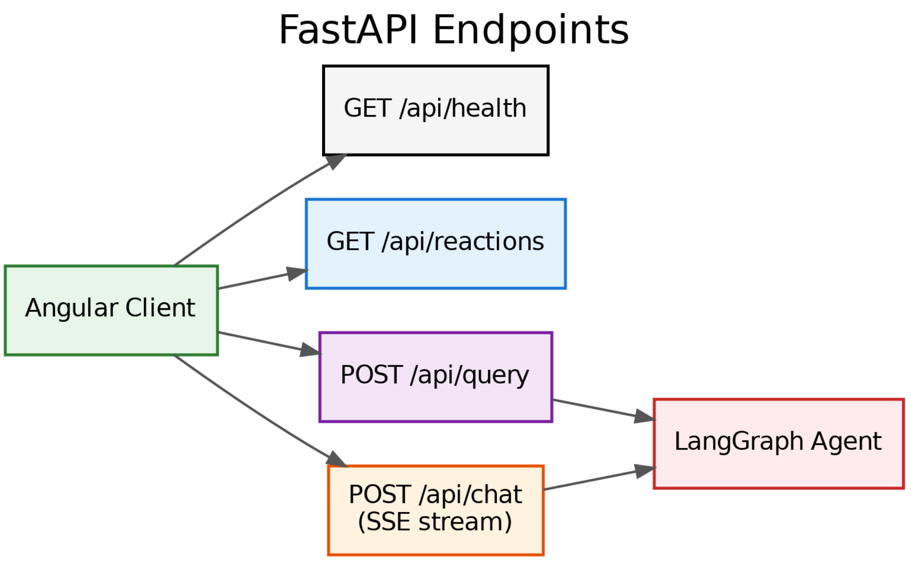

Figure 5. FastAPI endpoint structure.

7.1 Endpoint Specifications

Method	Path	Description	Response
GET	/api/health	Health check	{"status", "model", "region"}
GET	/api/reactions	List 13 reaction templates	{"reactions": [...]}
GET	/api/reactions/{id}/svg	Reaction SVG diagram	image/svg+xml
POST	/api/query	Full agent query (blocking)	All 6 expert analyses + final answer
POST	/api/chat	SSE streaming chat	status → answer → details → [DONE]
POST	/api/chat-with-files	SSE chat with file uploads	Same SSE protocol as /api/chat
POST	/api/analyze-smiles	RDKit molecular analysis	{"smiles", "scores", "svg"}

Table 10: API endpoint specifications.

7.2 SSE Stream Protocol (Updated)

The chat endpoint now emits status events for all 10 pipeline nodes and sends 6 expert fields in the details event:

```
data: {"type": "status","step": "rewrite_query","message": "Understanding..."}
data: {"type": "status","step": "router","message": "Classifying..."}
data: {"type": "status","step": "retrieve","message": "Retrieving..."}
data: {"type": "status","step": "reaction_expert","message": "Analyzing..."}
data: {"type": "status","step": "lipid_design_expert","message": "Evaluating..."}
data: {"type": "status","step": "generative_ai_expert","message": "Assessing..."}
data: {"type": "status","step": "property_prediction_expert","message": "Predicting..."}
data: {"type": "status","step": "literature_search","message": "Searching..."}
data: {"type": "status","step": "lead_agent","message": "Reasoning..."}
data: {"type": "answer","content": "...final answer..."}
data: {"type": "details","reaction_analysis": "...","lipid_design_analysis": "...",
"generative_analysis":"...","prediction_analysis":"...",
"literature_context":"...","web_context":"..."}
data: [DONE]
```

7.3 Reaction Templates

ID	Reaction	Reactants	Status
10001	Amide formation	Amine + Carboxylic acid	Valid
10003	Ester formation	Carboxylic acid + Hydroxyl	Valid
10005	Amine alkylation	Amine + Alcohol	Needs activation
10007	Thioether formation	Amine + Thiol	Valid
10009	Epoxide opening	Amine + Epoxide	Valid
10010	Michael addition (acrylate)	Amine + Alkyl acrylate	Valid
10011	Michael addition (acrylamide)	Amine + Alkyl acrylamide	Valid
10012	N-methylation	Amine + Methyl	Invalid
10013	Phosphate formation	Tert. amine + Dioxaphospholane	Valid
10014	Phosphate formation (alt)	Tert. amine + Dioxaphospholane	Valid
10015	Imine formation	Primary amine + Aldehyde	Valid
10016	Reductive amination	Secondary amine + Aldehyde	Valid
10017	Amide (reverse)	Primary amine + Aldehyde	Invalid

Table 11: All 13 reaction templates with validation status.

8. Frontend Application

The frontend is built with Angular 21 (standalone components) and Tailwind CSS v4. Version 2 updates include: conversation persistence via localStorage (up to 20 conversations), expanded expert detail panels showing all 6 agent outputs, file upload support for PDF/TXT/MD/CSV/DOCX attachments, and a revised workflow visualization reflecting the 10-node pipeline.

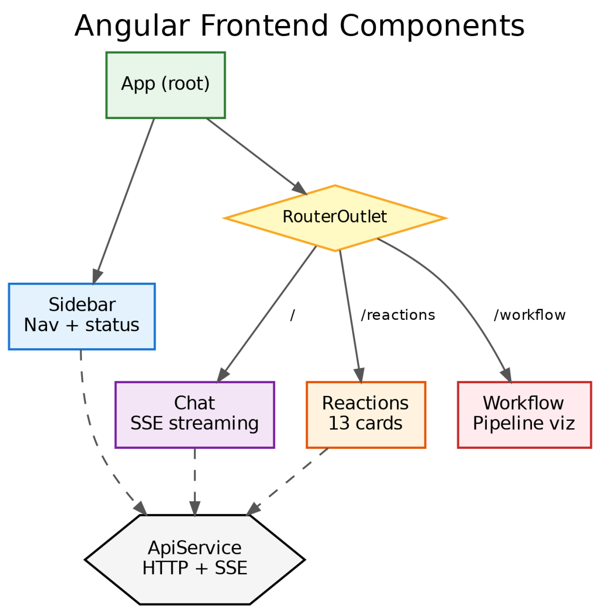

Figure 6. Angular component tree with services.

8.1 Component Summary

Component	Route	Key Features
App (root)	—	Shell layout: sidebar + router-outlet
Sidebar	—	Conversation list with new/delete/select, date grouping (Today, Yesterday, N days ago)
Chat	/	SSE streaming, file upload (PDF/TXT/MD/CSV/DOCX), 6 expert detail panels, markdown rendering
Reactions	/reactions	13 reaction cards with warning badges, SVG diagrams via backend API
Workflow	/workflow	10-step pipeline visualization with parallel execution indicators

Table 12: Frontend components.

8.2 Expert Detail Panels

Each assistant response includes an expandable 'Show expert analyses' section with 6 collapsible panels:

Panel	Source Agent	Content
■■ Reaction Analysis	reaction_expert	SMARTS matches, feasibility, conditions, competing reactions

Panel	Source Agent	Content
■■ Lipid Design	lipid_design_expert	Retrosynthesis routes, SAR insights, design rule compliance table
■■ Generative AI	generative_ai_expert	Model recommendations, reward functions, training plans
■■ Property Prediction	property_prediction_expert	ML models, uncertainty quantification, applicability domain
■■ Literature	literature_scout	PubMed articles with PMIDs, PubChem compound data
■■ Web Search	literature_scout	Web search results with titles, snippets, URLs

Table 13: Expert detail panels in the chat interface.

8.3 Conversation Management

The ConversationService provides localStorage-backed conversation persistence. Conversations are auto-titled from the first user message, sorted by recency, and limited to 20 entries. The sidebar displays conversations grouped by date (Today, Yesterday, N days ago). Each conversation stores the full message array including expert details, enabling review of past analyses.

9. Standalone and Desktop Applications

The standalone Dioxus WASM app and wry desktop app remain unchanged from v1. The Dioxus app provides focused molecular property analysis via RDKit, while the desktop app embeds the full Angular build as a native executable.

9.1 Deployment Comparison

Feature	Angular Web	Dioxus Standalone	Desktop (wry)
Runtime	Browser + Node.js	Browser only	Native (no browser)
Backend Dependency	Required (port 8000)	Required (port 8000)	Required (port 8000)
Build Size	~1 MB (dist/)	~200 KB (WASM)	~8 MB (binary)
Functionality	Full (chat + analysis)	Molecular analysis only	Full (chat + analysis)
Installation	npm install + ng serve	trunk build + Python	Single binary

Table 14: Deployment comparison.

10. Project Structure and Deployment

```
chem-agent/
■■■ .env # AWS Bedrock configuration (dual model)
■■■ pyproject.toml # Python dependencies (uv)
■■■ run.sh # Start both servers
■■■ data/
■ ■■■ papers/ # Research PDFs (3 + 30 files)
■ ■■■ lnp_data/ # Rules, reactions, CSVs
■ ■■■ faiss_lnp_index/ # Persisted FAISS index
■■■ src/
■ ■■■ agents/ # YAML agent definitions
■ ■ ■■■ __init__.py # Agent config loader
■ ■ ■■■ system.yaml # Master config + global constraints + dual model
■ ■ ■■■ router.yaml # Query classifier
■ ■ ■■■ reaction_expert.yaml # SMARTS template matching
■ ■ ■■■ lipid_design_expert.yaml # Retrosynthesis + SAR + rules
■ ■ ■■■ generative_ai_expert.yaml # Generation + RL
■ ■ ■■■ property_prediction_expert.yaml # ML prediction + UQ
■ ■ ■■■ literature_scout.yaml # PubMed/PubChem/web
■ ■ ■■■ lead_agent.yaml # Supervisor agent
■ ■■■ backend/
■ ■ ■■■ config.py # .env loader (MODEL_ID + FAST_MODEL_ID)
■ ■ ■■■ rag.py # FAISS + document ingestion
■ ■ ■■■ tools.py # PubMed, PubChem, web search
■ ■ ■■■ agent.py # LangGraph 10-node workflow (dual model)
■ ■ ■■■ main.py # FastAPI (7 endpoints)
■ ■■■ frontend/reactome-ui/ # Angular 21 project
■ ■■■ standalone/ # Dioxus WASM app
■ ■■■ standalone-desktop/ # wry desktop app
■■■ notes/ # Jupyter notebooks
■■■ docs/ # Technical documentation
```

10.1 Running the Application

```
# Backend (port 8000)
.venv/bin/uvicorn src.backend.main:app --host 0.0.0.0 --port 8000 --reload

# Frontend (port 4200) — requires Node.js 22+
cd src/frontend/reactome-ui && ng serve --host 0.0.0.0 --port 4200

# Or both at once
./run.sh

# Standalone Dioxus app (port 8001)
cd standalone && trunk build --release && python serve.py

# Desktop app (requires backend on port 8000)
cd standalone-desktop && ./build.sh && ./target/release/lnp-desktop
```

10.2 Remote Access via VS Code

- Forward ports **8000** and **4200** in the VS Code Ports panel
- Access frontend at **http://localhost:4200** in local browser
- Backend Swagger UI available at **http://localhost:8000/docs**

11. Changelog: v1 → v2 → v3

11.1 v2 → v3 Changes

Area	v2	v3
LLM: Router/Rewrite/Rerank	Claude Sonnet 4.5 (same as experts)	Claude 3.5 Haiku (fast, ~80% cheaper)
LLM: Expert Workers	Claude Sonnet 4.5	Claude Sonnet 4.5 (unchanged)
LLM: Lead Agent	Claude Sonnet 4.5	Claude Sonnet 4.5 (unchanged)
Config	Single BEDROCK_MODEL_ID	BEDROCK_MODEL_ID + BEDROCK_FAST_MODEL_ID
system.yaml	default_model only	default_model + fast_model
Health endpoint	{"model": "..."}	{"model": "...", "fast_model": "..."}
Workflow UI	No model attribution	Shows (Claude 3.5 Haiku) on fast nodes
Cost impact	Baseline	~80% reduction on 3 high-frequency nodes
Latency impact	Baseline	~20-30% reduction on router/rewrite/rerank

Table 16: Summary of changes from v2 to v3.

11.2 v1 $\rightarrow$ v2 Changes

Area	v1	v2
Agent pipeline	5-node, 3 experts	10-node, 6 agents (5 parallel workers + supervisor)
Agent config	Hardcoded prompts in agent.py	YAML files in src/agents/ with global constraints
Query routing	None (all queries $\rightarrow$ full pipeline)	Router classifies synthesis / lookup / general
Query rewriting	None	LLM rewrites using chat history for self-contained queries
Retrieval	Top-6 FAISS	Top-8 FAISS + LLM reranking $\rightarrow$ top-4
Expert: Design	Separate design_rules + synthesis_planner	Merged Lipid Design Expert (retrosynthesis + SAR + rules)
Expert: Gen AI	None	NEW: Generative AI Expert (generation + RL optimization)
Expert: Prediction	None	NEW: Property Prediction Expert (ML + uncertainty quantification)
External tools	None	PubMed, PubChem, web search (live API calls)
Lead Agent	Final answer node	Supervisor with conflict resolution + confidence levels
SSE details	3 fields	6 fields (all expert outputs)
Frontend details	3 panels	6 collapsible expert panels
Conversations	None (ephemeral)	localStorage persistence, sidebar management
File upload	None	PDF, TXT, MD, CSV, DOCX support via /api/chat-with-files
Workflow viz	6 steps (3 experts)	10 steps (5 parallel workers + supervisor)

Table 17: Summary of changes from v1 to v2.