



EUGENE PLATFORM

Project Evaluation

A Component-by-Component Assessment of the CSL Behring Biomedical Knowledge Graph & Agentic AI System

PREPARED FOR	CSL Behring — Business Development & R&D
PROGRAMME	Eugene Knowledge Graph & Agentic AI Platform
DATE	May 2, 2026
CLASSIFICATION	Confidential — Internal Use Only
VERSION	2.1 (Book Edition)

TABLE OF CONTENTS

FRONT MATTER

3

Foreword

3

How to Read This Document

3

Executive Summary

3

PART I — MISSION, STAKEHOLDERS & BUSINESS CONTEXT

4

Mission and User Personas

4

Stakeholder Map

5

Business Value and Use Cases

6

PART II — SYSTEM ARCHITECTURE

6

Architectural Overview

6

End-to-End Request Flow

8

PART III — COMPONENT DEEP-DIVES

10

Streamlit Chat UI (eugene-agent-ui)

10

Agent Backend (eugene-agent-ws)

11

MCP Server (eugene-mcp)

12

Core API (eugene_ws)

13

Neo4j Layer

14

Data Ingestion and GraphRAG

15

PART IV — CROSS-CUTTING CONCERNS

16

Authentication and Authorisation

16

Performance Profile

18

Security Posture

19

Code Quality and Engineering Discipline

19

Observability

19

Operational Runbook

20

PART V — DEPLOYMENT AND ENVIRONMENTS

21

AWS Topology

21

Local Development

22

PART VI — RISK, CONCLUSION AND APPENDICES

22

Documented Risk Register 22

Programme-Level Conclusion 23

Appendix A — Glossary 24

Appendix B — Core API Endpoint Inventory 24

Appendix C — MCP Tool Inventory 25

Appendix D — Source Documents Consulted 26

FRONT MATTER

Foreword

This volume presents a complete, component-by-component evaluation of the Eugene biomedical knowledge graph and agentic AI platform commissioned by CSL Behring. The intent is twofold. First, to record — at a single source of truth — what has been built: the architecture, the services, the data, the deployment topology, the security posture, the operational practices. Second, to present an honest, systematic assessment of each constituent component, so that programme leadership and engineering teams alike can make informed decisions about what to harden, what to extend, and what to retire as Eugene transitions from first-generation proof of value to a strategic CSL asset.

The material in these pages has been distilled from four primary sources: the Eugene Project Validation Document (v1.0, March 2026); the Eugene Stage 1 Assessment & Audit Report (April 2026); the Eugene Developer Guide; and the canonical architecture diagram preserved in ``docs/architecture/eugene-architecture.drawio``. Where individual claims rest on a specific source, the inline citation tags from those documents are preserved verbatim. Where the underlying material was thin, this volume defers to silence rather than speculation.

The companion volume, **Eugene Platform — Technical Recommendations & Next-Generation Architecture**, sets out remediation, the proposed multi-agent target architecture, and a programme of user-experience enhancements driven by the findings recorded here. The two documents are intended to be read together.

How to Read This Document

The volume is organised in six parts:

- Part I — Mission, Stakeholders, and Business Context — establishes why Eugene exists, for whom, and against which decision-making workflows it is benchmarked.
- Part II — System Architecture — surveys the platform at the topology level: tiers, services, ports, technologies, and the canonical request flow.
- Part III — Component Deep-Dives — assesses each major service in turn, with sections for design, implementation, current state, evaluation, and component-specific findings.
- Part IV — Cross-Cutting Concerns — addresses authentication, security, performance, code quality, observability, and operations as horizontal concerns.
- Part V — Deployment and Environments — covers AWS topology, environment matrix, Terraform module structure, and local-development practice.
- Part VI — Risk, Conclusion and Appendices — consolidates the documented risk register, draws programme-level conclusions, and provides reference appendices (API endpoint inventory, MCP tool inventory, and a glossary).

Each component chapter follows a consistent shape: Purpose, Design, Implementation, Current State, Evaluation, and Findings. The Findings sub-section ties back to the 21 issues identified in the Stage 1 audit so that the link between observed behaviour and remediation work is explicit.

■ *This document is confidential to CSL Behring. Reproduction or distribution outside CSL Behring requires programme-lead approval. Where individual usernames, credentials, or account identifiers appear, they are illustrative only; treat all such values as sensitive.*

Executive Summary

What Eugene Is

Eugene is an enterprise-grade, AI-powered biomedical knowledge graph platform purpose-built for CSL Behring's competitive intelligence and research analytics operations. It allows non-technical users to pose complex, multi-hop biomedical questions in plain English and to receive synthesised, evidence-backed answers in seconds. The underlying graph integrates authoritative public data — USPTO patents, PubMed literature, ClinicalTrials.gov records — with internal CSL data assets, delivering a single conversational surface across what was previously a fragmented set of databases and tools.

What Eugene Achieves Today

The platform answers questions such as "Which organisations hold patents on drugs targeting the same indication as our lead compound, and what is their clinical trial status?" or "Find all organisations in Germany with active Phase III trials in rare disease" — answerable by a non-technical user in under thirty seconds. The deployment is functionally complete: five services run end-to-end across two AWS environments (us-east-1 development; eu-central-1 production) plus a Docker Compose local stack; authentication is wired through Microsoft Entra ID; and twelve schema-validated MCP tools mediate the agent's access to a graph of approximately 484,000 nodes and twenty-one million edges held in Neo4j 5.26 Community Edition.

Architectural Strengths

The Stage 1 audit recognised six structural strengths that should be preserved through any future refactor: clean Domain-Driven Design with hexagonal layering in the Core API; a comprehensive ontology spanning forty-four node types and thirty-one relationship types; dual-LLM provider auto-detection (OpenAI and Anthropic); a stateless MCP server design that scales horizontally; token-by-token streaming via Server-Sent Events for ChatGPT-class user experience; and parameterised Cypher queries with a hard cap on traversal depth, preventing both injection attacks and pathological graph fan-out.

What Is Missing

Three categories of gap recur across this assessment. The first is operational and security hardening required before broad enterprise rollout: SSL verification must be re-enabled in MCP-client connections; the shared conversation manager must be instanced per-session to remove a known race condition; query timeouts, rate limits, structured logging, and distributed tracing must be implemented. The second is data and feature gaps: USPTO and PubMed counts in the production graph are presently zero (deletion without reload, and abandoned partial ingestion respectively); RBAC at the row level is unimplemented; the GraphRAG pipeline is deprecated. The third is observability: there is no LLM cost instrumentation, no structured JSON logging, and only five of twenty Core API endpoints are covered by canaries.

Position Heading Into Stage 2

Eugene is a credible first-generation platform with sound architecture and demonstrated value delivery. It is ready for limited stakeholder use today. Promotion to broad enterprise rollout is a programme of work — sized at approximately eight engineering weeks for the security and operational backbone, and a further twelve to sixteen weeks for the next-generation multi-agent topology described in the companion document. The investment is well-targeted: Eugene's current architectural strengths mean that hardening work compounds rather than displaces what already works.

PART I — MISSION, STAKEHOLDERS & BUSINESS CONTEXT

Mission and User Personas

1.1 Mission Statement

Eugene exists to compress the time required to answer biomedical competitive-intelligence and research questions from days of manual database navigation to seconds of conversational interaction. The Project Validation Document positions Eugene as "an enterprise-grade, AI-powered biomedical knowledge graph platform purpose-built for CSL Behring's competitive intelligence and research analytics operations." The platform draws on authoritative public sources — USPTO, PubMed, ClinicalTrials.gov — alongside internal CSL data assets, with the explicit objective that a non-technical user should pose a multi-hop biomedical question in plain English and receive a synthesised, evidence-backed answer in real time.

1.2 Primary User Personas

The platform serves three primary personas. Each shares two needs that legacy database UIs do not satisfy: synthesis across heterogeneous sources, and conversational follow-up that preserves context across turns.

1.2.1 Business Development Analysts

BD analysts evaluating asset-in-licensing opportunities require depth across therapeutic areas, deal precedent, sponsor concentration, and the patent landscape surrounding a candidate asset. Eugene allows the analyst to express those questions conversationally — "What other companies have late-stage assets targeting Factor IX?" — and receive a structured response with citation back to graph nodes and relationships.

1.2.2 R&D; Scientists

Research scientists exploring target rationale, mechanism-of-action precedent, and trial outcomes across modalities use Eugene as a discovery tool. The graph exposes typed relationships (Drug-TARGETS-GeneProtein, ClinicalTrial-STUDIES-Drug, GeneProtein-ASSOCIATED_WITH-Disease) that allow the agent to traverse evidence chains rather than return isolated keyword matches.

1.2.3 Medical Affairs

Medical Affairs professionals tracking real-world evidence, indication overlap, and competitor pipelines benefit from the platform's ability to surface patent-trial-publication networks around a single anchor entity. The conversational surface materially lowers the friction of this workflow relative to a stack of separate database UIs.

1.3 Sample User Questions

The Project Validation Document records the following representative questions, all of which the platform demonstrably answers end-to-end today:

- Which organisations hold patents on drugs targeting the same indication as our lead compound, and what is their clinical trial status?
- Find all organisations in Germany with active Phase III trials in rare disease.
- Which drugs target EGFR and what trials are sponsored by which organisations?
- Trace the connection between AstraZeneca and the KRAS gene through patents and trials.
- What are the relationships of Hemophilia A in the graph?

Stakeholder Map

2.1 Roles and Responsibilities

The Project Validation Document records the following stakeholder map, which is reproduced here as the canonical reference for ownership boundaries:

Role	Stakeholder Group	Responsibility
Executive Sponsor	CSL Behring Leadership	Strategic direction; budget approval
Product Owner	R&D; / Competitive Intelligence	Requirements prioritisation; user acceptance
Platform Engineering	AI / Data Engineering Team	Architecture, development, deployment
End Users	Scientists, Analysts, Medical Affairs	Query, interpret, and act on insights
Security / Compliance	IT Security; Legal	Authentication, data governance review
Infrastructure	Cloud Platform Team	AWS environment management

2.2 Decision Cadence

Programme-level decisions follow a quarterly cadence with monthly steering. Component-level engineering decisions are taken inside the Platform Engineering team. The Stage 1 audit observed that this cadence is appropriate for the current scale, but recommends a Change Advisory Board step for the production (eu-central-1) environment once the platform is opened to a broader user base, to manage release blast radius.

Business Value and Use Cases

3.1 Value Proposition

The validation document articulates five value drivers: accelerated competitive intelligence (analysts query drug pipelines in seconds rather than hours); reduced research latency (the agent eliminates manual Cypher authorship); centralisation of biomedical knowledge into a single navigable graph; scalable analytics infrastructure based on cloud-native services; and auditability via structured MCP tool calls that produce explainable, traceable reasoning chains.

3.2 Indicative Workflows

Three workflows recur in the audit material and are reproduced here as evidence that the platform's design fits its intended uses:

1. Single-entity lookup. Resolve "Imatinib" to a canonical drug node, then fetch details — mechanism of action, status, approval date, aliases. Two MCP tool calls; sub-second to first token.
2. Multi-hop competitive intelligence. From an EGFR anchor, identify EGFR-targeting drugs, then for each drug trace clinical trials, then filter by sponsor country. Six to ten MCP tool calls; six to twelve seconds end-to-end.
3. Path-based reasoning. Given two named entities (an organisation and a gene), first establish reachability, then return the actual paths through patent-ownership and drug-targeting edges. Four MCP tool calls.

PART II — SYSTEM ARCHITECTURE

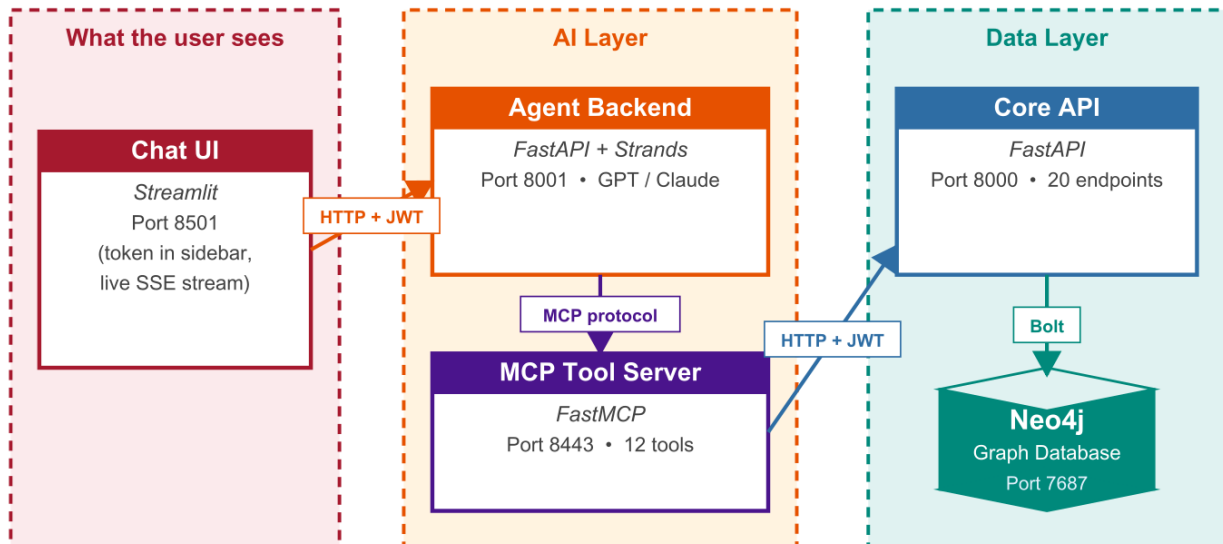
Architectural Overview

4.1 Architectural Style

Eugene implements a four-tier microservice architecture following Domain-Driven Design principles with a hexagonal (ports-and-adapters) internal structure inside the Core API. The five running services — User Interface, Agent Backend, MCP Server, Core API, and Neo4j — are containerised, communicate over well-defined HTTP and Bolt protocols, and run on Docker Compose locally and AWS ECS Fargate in cloud environments.

Eugene Platform — High-Level Architecture

(per docs/DEVELOPER_GUIDE.md — flowchart LR)



4.2 Layer Responsibilities

Tier	Service	Responsibility
Presentation	eugene-agent-ui	User interaction; live token rendering
Agent	eugene-agent-ws	Autonomous reasoning; tool selection; LLM inference
Tool / Orchestration	eugene-mcp	Structured, authenticated tool surface for agent
Data API	eugene_ws	Canonical graph data API; 20 routers
Persistence	Neo4j 5.26 CE + Milvus	Graph + vector storage

4.3 Why Microservices, Why Hexagonal

The microservice split was driven by three forces. First, independent scaling: the agent tier is bound by LLM throughput, the MCP tier by tool invocation rate, the API tier by Cypher concurrency, and Neo4j by working-set memory; coupling these into a monolith would force the slowest tier to dictate cluster sizing. Second, isolation of failure domains: an LLM provider outage degrades the agent tier without taking down the read-API for direct Cypher queries. Third, deployability: each service has its own Dockerfile, its own ECR repository, and its own ECS task definition, allowing rolling updates without coordinated downtime.

Hexagonal layering inside the Core API — Router, Orchestrator, Provider, Mapper, Adapter — keeps domain logic free of transport and persistence concerns. Each layer has one reason to change. The audit identified this as a structural strength worth preserving through any future refactor.

4.4 Technology Inventory

Technology Stack — Layered Inventory

Frontend	Streamlit	Next.js + TypeScript	Cytoscape.js	Chakra UI		
Agent	Strands (ReAct)	Anthropic Claude	OpenAI GPT-4.1	FastMCP client		
Tool / API	FastMCP	FastAPI	Pydantic v2	httpx / aiohttp		
Data	Neo4j 5.26 CE	APOC	GDS	Milvus	sentence-transformers	
DevOps	Docker	ECS Fargate	ECR	Terraform	GitHub Actions	
Quality	Black	Ruff	isort	Pytest	MLflow	pre-commit

The diagram above arranges Eugene's stack by tier; the same information is reproduced in tabular form below for procurement and audit reference.

Concern	Technology
Language	Python 3.12 / 3.13
UI	Streamlit (v1) + Next.js + TypeScript (v2)
Agent framework	Strands (ReAct pattern)
LLM	OpenAI GPT-4.1 or Anthropic Claude Sonnet (auto-detect)
API framework	FastAPI
Validation	Pydantic v2
Tool protocol	FastMCP (HTTP, stateless_http=True)
Graph DB	Neo4j 5.26.9 Community + APOC + GDS
Vector store	Milvus + sentence-transformers
HTTP clients	httpx, aiohttp
Containerisation	Docker, Docker Compose
Orchestration	AWS ECS Fargate
IaC	Terraform
Code quality	Black, Ruff, isort, pre-commit
Tests	Pytest with parallel execution

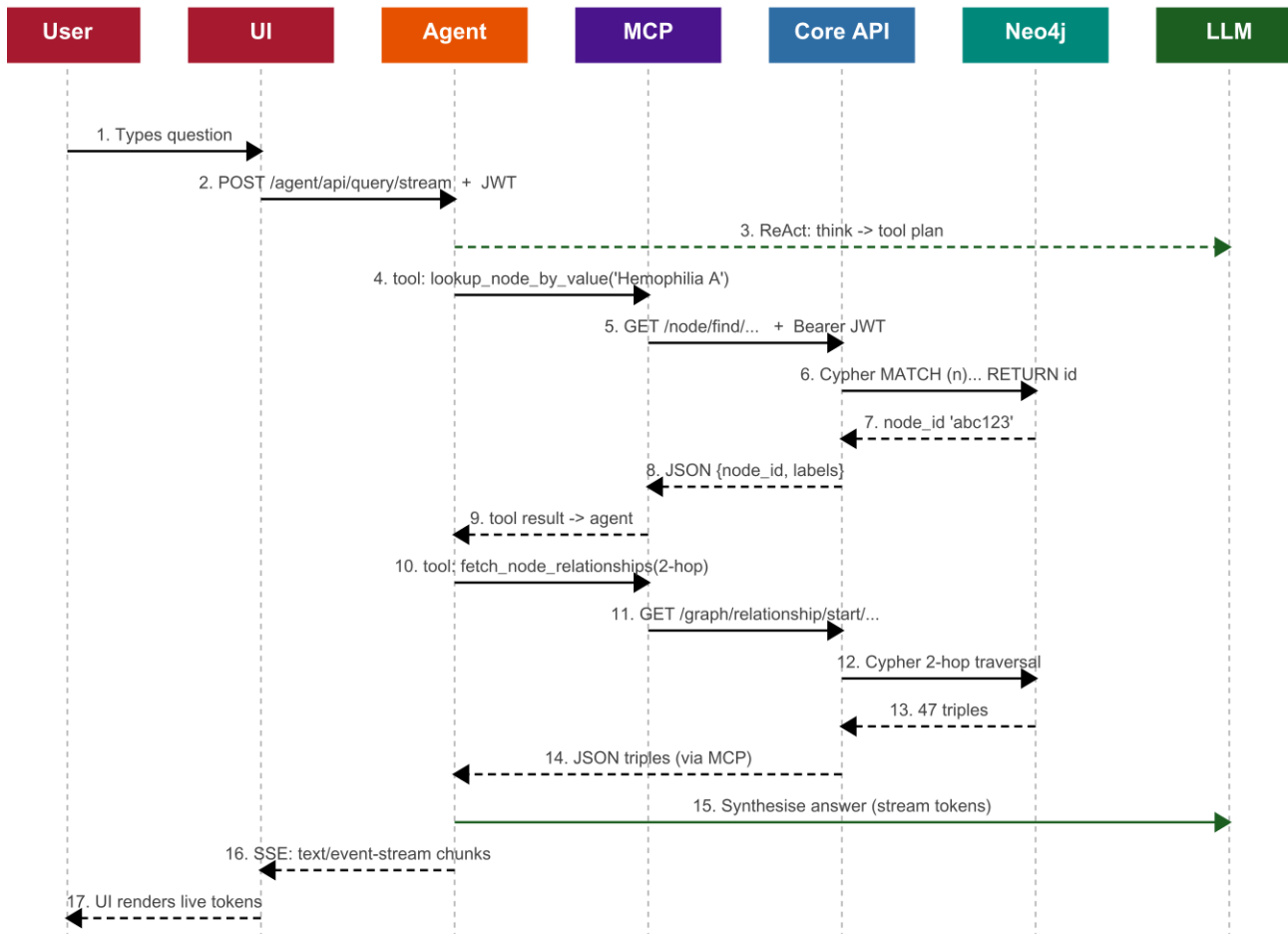
End-to-End Request Flow

5.1 Canonical Sequence

When a user submits a question, control flows through every tier of the platform. The diagram below traces the canonical seventeen-step path for the example query "What are the relationships of Hemophilia A?". Three properties are worth highlighting for stakeholders: the user sees partial answers within roughly one second through SSE streaming; the same Eugene-issued JWT propagates from browser through agent,

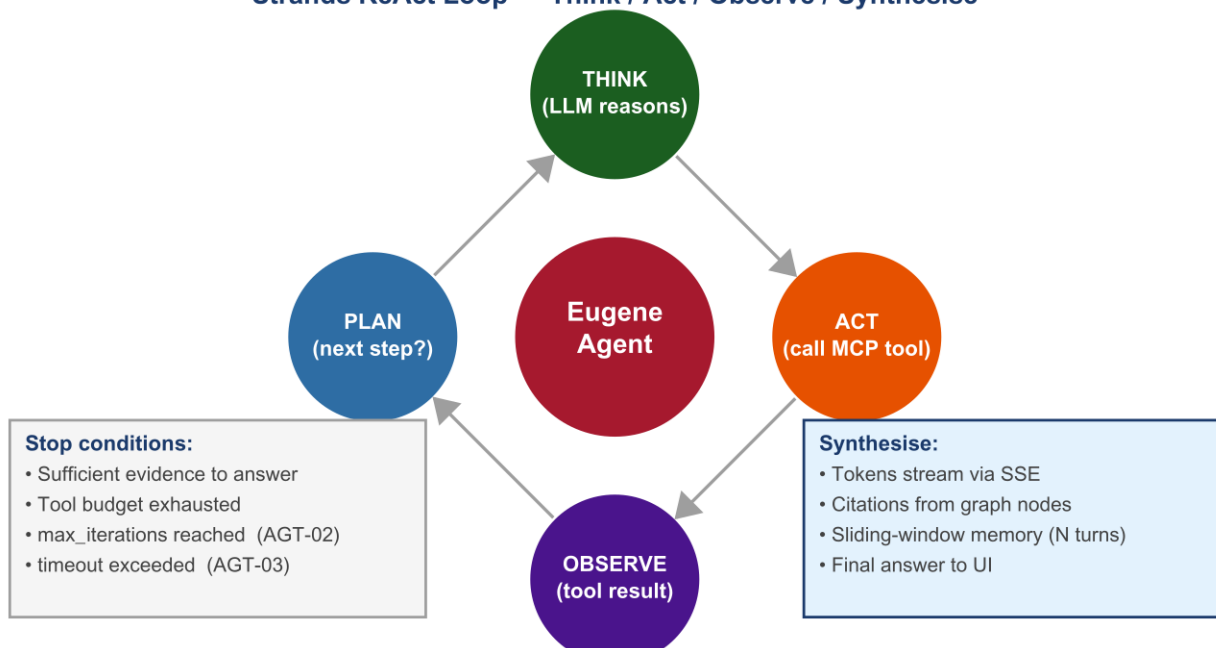
MCP, and Core API, with each tier independently validating signature, audience, issuer, and expiry; and Cypher traversal depth is hard-capped at two hops, so the LLM cannot induce arbitrary database queries.

End-to-End Query Sequence — User question to streamed answer



5.2 ReAct Loop Inside the Agent

Strands ReAct Loop — Think / Act / Observe / Synthesise



Inside the Agent Backend, the Strands framework drives a ReAct loop:

1. Think — the LLM produces a reasoning step and decides which tool to call.
2. Act — the agent invokes the chosen MCP tool with structured arguments.
3. Observe — the tool result is returned and injected into the agent's context.
4. Repeat — until the agent has sufficient evidence or its tool budget is exhausted.
5. Synthesise — the agent emits a token-by-token answer over Server-Sent Events.

The agent's stopping criterion is described in the validation document as "sufficient evidence to produce a complete, cited answer, or when the tool call budget is reached." The audit identifies a finding (AGT-02) that the iteration limit is not currently bounded in code; this is addressed in the companion document under the recommended quick wins.

5.3 JWT Propagation

A single Eugene-issued HS256 JWT travels with the request from the browser through every tier. Each tier independently validates signature, audience, issuer, and expiry. The MCP server, on receiving a tool call, forwards the same Bearer token to the Core API; the Core API validates again before any Cypher is issued. This design ensures that future row-level RBAC policies are honoured automatically; the platform does not have a backend service identity that bypasses user context.

PART III — COMPONENT DEEP-DIVES

Streamlit Chat UI (eugene-agent-ui)

6.1 Purpose

The Streamlit Chat UI is the conversational entry point to Eugene. It accepts a natural-language question, posts it to the Agent Backend with a Bearer JWT, and renders the streamed response token by token. The implementation lives at ``agents/eugene-agent-ui/src/eugene_agent_ui.py``; it runs locally on port 18501 and on cluster port 8501 inside the container.

6.2 Design

Streamlit was chosen because it allowed a working chat surface to be delivered in Python alongside the rest of the stack, reusing the same SSE client patterns as the Agent Backend's tests. The state model is conventional Streamlit: session-state holds the conversation identifier, message list, and JWT; each user input triggers a re-render that appends the assistant turn as it streams.

6.3 Implementation

Concrete features observed in the source:

- Sidebar with conversation ID tracking and a 'New Conversation' button.
- Datasource checkboxes (eugene, pubmed, http) that toggle which categories of MCP tools the agent is permitted to invoke for the next turn.
- OAuth token input via a masked text field in the sidebar; the token persists in session state until the user explicitly clears it.
- A greeting prompt rendered the first time a token is supplied, providing context to first-time users.
- Chat message history rendered using Streamlit's `chat_message` component.
- Suggestion pills — seven curated example queries that one-click populate the input box, intended to scaffold first-use.
- Streaming response loop using `asyncio` over an SSE endpoint, with a trailing block-cursor character (█) while tokens are still arriving.

6.4 Current State

The UI works end-to-end. JWTs round-trip correctly. Streaming is smooth on broadband connections, with first-token latency dominated by LLM provider round-trip rather than network or rendering. The seven suggestion pills cover drug aliases, organisation queries, multi-hop relationship queries, PubMed lookups, and graph-fact extraction — a representative cross-section of the platform's strengths.

6.5 Evaluation

The Streamlit choice has accelerated initial delivery but constrains the user experience that can be offered. Specifically: the absence of citation panels, evidence drilldown, graph visualisation, and conversation export are all artefacts of the framework's batteries-included assumption set rather than fundamental gaps in the platform's data layer. The audit notes a Next.js v2 implementation under `agents/eugene-agent-ui-next/` that addresses these constraints with a three-pane layout (sidebar / chat / interactive Cytoscape graph), floating glass-card overlays, and a node details panel — but this v2 surface is not yet promoted to production.

6.6 Findings Linkage

No Stage 1 findings are filed against the Streamlit UI directly; the issues above are characterised as enhancement opportunities rather than defects. The companion document's UI-enhancement chapter proposes fifteen concrete improvements that, taken together, materially raise the surface area of the platform without altering the underlying data path.

Agent Backend (eugene-agent-ws)

7.1 Purpose

The Agent Backend hosts the autonomous reasoning loop that turns a user question into a sequence of tool calls and a synthesised answer. It is a FastAPI service that exposes a single user-facing route — `POST /agent/api/query/stream` — and an internal `/health` endpoint.

7.2 Design

The agent uses the Strands framework, configured with a domain-specific system prompt that positions the agent as a biomedical competitive-intelligence expert. It is initialised per-conversation with a sliding-window memory of the most recent N turns, registers twelve MCP tools (plus locally available utilities such as a calculator and HTTP request tool), and runs in fully async, streaming mode.

7.3 Implementation Highlights

- ReAct iteration is driven by Strands; the agent does not execute Cypher directly.
- MCP tools are loaded at conversation start via `MCPClient.list_tools_sync()` against the configured eugene-mcp endpoint.
- LLM provider is auto-detected: if `ANTHROPIC_API_KEY` is set the Anthropic client is used; otherwise the OpenAI client. Both can be pinned via `ANTHROPIC_MODEL_ID` / `OPENAI_MODEL_ID` environment variables.
- Token stream emission uses FastAPI's `StreamingResponse` with `text/event-stream`.
- Per-conversation state is stored via a `FileSessionManager`, which writes session checkpoints to local disk inside the container.

7.4 Current State

The agent successfully drives multi-turn conversations through twelve to sixteen tool calls in the worst observed case, reliably terminating with a synthesised answer that cites the graph entities consulted.

Concurrent users are supported on local Docker (five to ten) and in ECS production (fifty to two hundred, depending on task count and instance sizing).

7.5 Evaluation

The agent's design is correct in its choice of framework, prompt construction, and tool-mediated boundary. However, the audit identified six findings filed against the agent tier, four of which are HIGH severity and one CRITICAL. These are reproduced verbatim below and addressed in the companion document.

7.6 Component Findings

Finding	Severity	Description
AGT-01	CRITICAL	Shared SlidingWindowConversationManager — race condition under concurrent load; sessions can corrupt each other.
AGT-02	HIGH	Unbounded ReAct loop — no max_iterations; agent can cycle indefinitely on tool calls.
AGT-03	HIGH	No agent execution timeout — streaming connection held until the client times out (typically 120 seconds).
AGT-04	HIGH	Anthropic client timeout not configured — only OpenAI has a 60-second timeout; Anthropic has none.
AGT-05	HIGH	python_repl tool always loaded — the LLM has access to arbitrary code execution unless explicitly disabled.
AGT-06	MEDIUM	FileSessionManager — no recovery, no cleanup, file contention under concurrent writes for the same conversation.

MCP Server (eugene-mcp)

8.1 Purpose

The MCP Server is the governed abstraction layer between the agent and the Core API. The agent never queries Neo4j or the Core API directly; every tool invocation passes through the MCP layer, which applies authentication, schema validation, and audit logging.

8.2 Design Principles

- Tool governance — the agent can only perform operations explicitly defined and approved as MCP tools.
- Authentication propagation — every tool call carries and validates the user's JWT, ensuring agent actions respect user permissions.
- Abstraction — the agent is shielded from Core API URL structure, Cypher syntax, and data transformation logic.
- Auditability — all tool calls are structured and loggable at the MCP layer.
- Statelessness — FastMCP is configured with stateless_http=True, enabling horizontal scale-out without sticky sessions.

8.3 The Twelve Tools

Tool	Description	Maps to API
fetch_identity	Resolve a name to canonical node ID	/auth/whoami, /node/find/{value}
fetch_by_label	List nodes by label type	/labels/{label}
fetch_similar	Find similar nodes by embedding	/similarity/{label}
lookup_node_by_value	Find a node by name (fuzzy match optional)	/node/find/{value}
fetch_node_details	Retrieve full property set for node IDs	/node/details (POST)
fetch_drug_aliases	Return all known synonyms for a drug	/drugs/aliases/{name}
fetch_facts	Extract human-readable triples from relationships	/graph/facts/start/{id}
fetch_node_relationships	Return typed edges from a node (1- or 2-hop)	/graph/relationship/start/{id}
fetch_paths	Return path(s) connecting two nodes	/graph/path/start/{id}/end/{id}
has_reachable_path	Boolean reachability check	/graph/reachability/start/{id}/end/{id}
find_organization_names	Search organisations by name, country, domain	/organizations/{pattern}
find_organization_assets	Drugs, trials, and IP for an organisation	/organizations/assets/{id}

8.4 Evaluation

The MCP layer is one of the strongest design decisions in the platform. By interposing a tool surface between the agent and the underlying API, Eugene achieves audit, abstraction, and access control without compromising the agent's reasoning flexibility. The audit recognised this as a positive structural finding.

8.5 Component Findings

One CRITICAL finding (SEC-01) is filed against the MCP client: SSL verification is disabled (`verify=False`) on the `httpx` connection from the Agent Backend to the MCP server, exposing the in-cluster traffic to man-in-the-middle attack. The remediation is a half-day fix, addressed in Phase 1 of the recommended programme.

Core API (eugene_ws)

9.1 Purpose

The Core API is the canonical data interface to the Eugene knowledge graph. It is a FastAPI application exposing twenty REST routers organised by biomedical domain, all backed by Cypher against a single Neo4j database.

9.2 Hexagonal Layering

Each request flows through a strict five-layer pipeline: Router accepts the HTTP call and validates the request body using Pydantic v2; Orchestrator (optionally) composes multiple Providers for complex workflows; Provider executes a single business operation, calling Adapter and Mapper; Mapper transforms a DataFrame into a domain model and on into a response model; Adapter constructs the parameterised Cypher query and executes it against Neo4j. Domain logic remains independent of transport and persistence.

9.3 Endpoint Inventory

Domain	Routers
Foundation	label, n_hop, facts, node_details, count
Identity	node_id_lookup, similarity, facet
Drug	drug_alias_search
Organisation	organization_search
Patent	patent_search, patent_count
PubMed	pubmed_search, pubmed_count
Path / Graph	search_path, n_hop
Stats	database_stats
Auth & system	auth, root, health, release_notes

9.4 Cypher Discipline

All twenty routers use parameterised Cypher with \$variable binding; no string interpolation appears anywhere in the source. The MAX_SUPPORTED_HOPS constant, set to two, is enforced inside the n-hop adapter and prevents the graph fan-out that would otherwise occur at depth three or greater. The audit recognised this discipline as a structural strength.

9.5 Evaluation

The Core API is the most mature element of the platform. Twenty endpoints, all schema-validated, all documented through Swagger at /docs, all backed by parameterised Cypher. The DDD/hexagonal architecture has paid dividends in testability and consistency.

9.6 Component Findings

Finding	Severity	Description
DB-01	HIGH	No Neo4j query timeout — slow queries block adapter threads indefinitely.
DB-02	MEDIUM	No explicit connection-pool sizing; defaults limit concurrent throughput.
DB-03	MEDIUM	Deep pagination has no offset cap; very large OFFSET values induce slow scans.
SEC-02	HIGH	Allowlist raises generic Exception (HTTP 500) instead of HTTPException 403.
SEC-03	HIGH	No rate limiting on any endpoint; cost and DoS exposure.

Neo4j Layer

10.1 Purpose

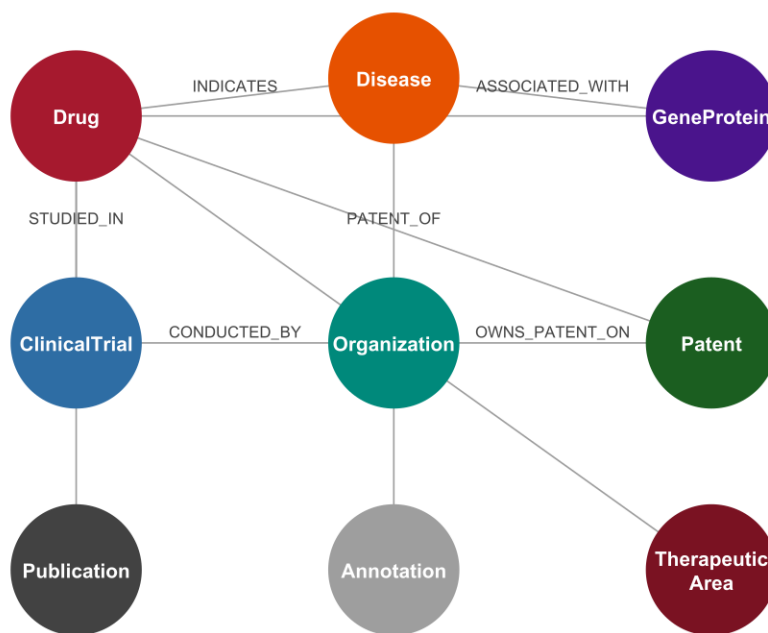
Neo4j is the canonical persistence for Eugene. It holds approximately 484,000 nodes across nine labels and approximately twenty-one million typed, directed edges. Cypher is the only query language used; APOC and GDS plugins are bundled for path analysis, similarity, and bulk import.

10.2 Memory Tuning by Environment

Environment	Pagecache	Heap initial	Heap max
Local (Docker)	512 MB	512 MB	1 GB
Development (AWS)	4 GB	2 GB	4 GB
Production (AWS)	16 GB	4 GB	8 GB

10.3 Schema Recap

Eugene Knowledge Graph — Schema (9 node labels, principal edges)



Approx. 484k nodes • 21M edges • Bolt protocol • parameterised Cypher only

The graph carries nine principal node labels — Drug, Disease, GeneProtein, ClinicalTrial, Patent, Organization, Publication, Annotation, and Therapeutic Area. The principal relationship types are TARGETS, INDICATES, ASSOCIATED_WITH, OWNS_PATENT_ON, CONDUCTED_BY, STUDIES, PUBLISHED_IN, SYNONYM_OF, and PARENT_OF. The full data table is reproduced in Appendix C.

10.4 Evaluation

Neo4j is the right tool for this workload — relationship-rich biomedical data with multi-hop queries that map cleanly onto Cypher's pattern syntax. The MAX_SUPPORTED_HOPS=2 cap is a pragmatic guard against fan-out and should be preserved. The Community Edition limits available index types and clustering modes; promotion to Enterprise is not currently planned but should be evaluated as the user base expands.

10.5 Operational Issues

- Neo4j on production is provisioned manually on EC2; it is not yet automated through Terraform.
- Neo4j SSL certificate rotation is a manual operational step.
- There is no automated S3 snapshot policy; documented as risk TR-07.
- Heavy analytics queries can occasionally provoke an instance restart; documented in the maintenance runbook.

Data Ingestion and GraphRAG

11.1 Pipeline Overview

Data ingestion is a three-step pipeline: Download & Parse (USPTO, PubMed, ClinicalTrials.gov downloads, plus LLM-assisted entity and organisation name extraction from PDFs); Transform (JSON to text, then into Entity and Relationship dataclasses); Load (MERGE-based Cypher ingestion via Neo4jGraphragAdapter and store_triples.py).

11.2 Data Sources Configured

- USPTO Patents — patent records, filing dates, assignees.
- PubMed — journal articles, abstracts, citations.
- ClinicalTrials.gov — trial records, phases, sponsors.
- Internal CSL data — proprietary records.
- DrugBank — drug mechanisms and indications.
- Organisation database — pharma and biotech companies.

11.3 Current Loading Status

■ *USPTO patents count is currently zero in production (data deleted, never reloaded). PubMed count is currently zero (partial ingestion abandoned). Therapeutic-area subgroup counts are currently zero. These gaps are documented in the maintenance runbook and addressed in the companion document's data-readiness backlog.*

11.4 GraphRAG

The GraphRAG NLP-extraction pipeline is documented as deprecated. The original concept — extracting biomedical triples directly from unstructured literature using LLM-assisted entity recognition — remains attractive but the current implementation has been frozen. The next-generation architecture in the companion document reframes this capability through Amazon Bedrock Knowledge Base, decoupling unstructured retrieval from the graph.

11.5 Findings

Finding	Severity	Description
GRAPH-01	MEDIUM	No graph statistics dashboard surfacing live counts or growth.
GRAPH-02	MEDIUM	Graph data freshness not tracked; no last_updated property.

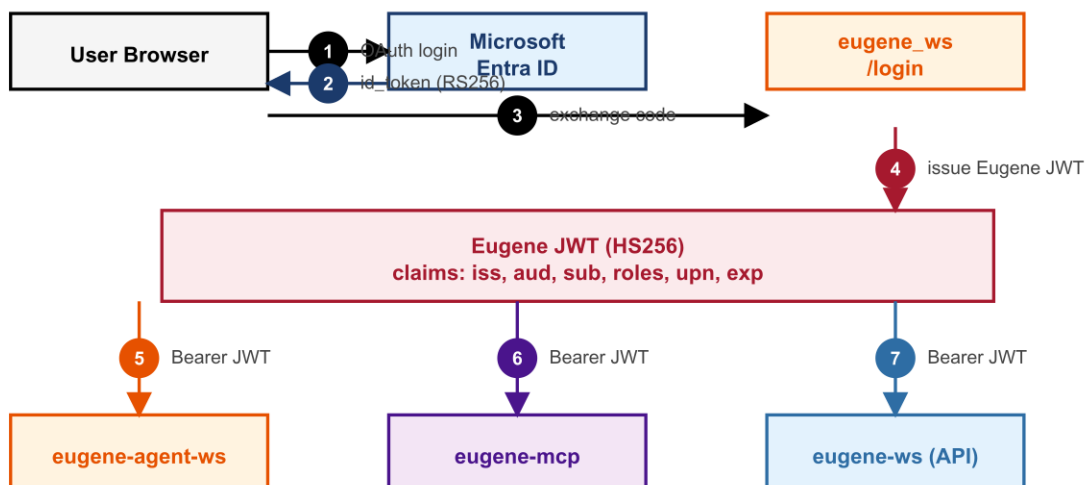
PART IV — CROSS-CUTTING CONCERNS

Authentication and Authorisation

12.1 Two-Layer Identity Model

Eugene operates a two-layer identity model. Layer 1 is enterprise SSO via Microsoft Entra ID using OAuth 2.0 Authorization Code flow; Layer 2 is an Eugene-issued JWT that propagates between services. The two layers are necessary because Entra ID issues tokens scoped to a tenant and audience that the downstream services do not directly own; eugene_ws acts as an issuer for service-scoped tokens that include the additional claims (roles, allowlist membership) required by the platform.

Authentication Flow — Entra ID → Eugene JWT → Service Mesh



Local mode (ENVIRONMENT=local) bypasses steps 1–3 and issues a development JWT directly.

12.2 Eugene JWT Claims

```

{
  "iss": "https://eugene.ai.cslg1.cslg.net/{tenant_id}",
  "aud": "api://eugene/{client_id}",
  "sub": "eugene.test",
  "upn": "eugene.test@cslbehring.com",
  "roles": ["user.public.read"],
  "exp": 1711086400
}
  
```

12.3 Authorisation Boundaries

Boundary	Mechanism	Enforcement Point
User → eugene-agent-ui	Entra ID SSO	Browser redirect
UI → eugene-agent-ws	Eugene JWT	FastAPI dependency
Agent → eugene-mcp	Eugene JWT (forwarded)	FastMCP middleware
MCP → eugene_ws	Eugene JWT (forwarded)	FastAPI dependency

12.4 Allowlist and Local Mode

An EUGENE_AGENT_ALLOWLIST environment variable (pipe-separated UPNs) gates agent invocation. Empty allowlist permits all authenticated users. Local development bypasses Entra ID via ENVIRONMENT=local; the agent issues a development JWT directly. This configuration must not be enabled outside developer environments.

12.5 RBAC Status

Row-level RBAC is not implemented today. The roles claim is present in the JWT but not enforced inside Cypher queries. Future work scoped to Q3 2026 introduces Neo4j property-based access control for confidential data; until that lands, only public data should be ingested into the graph.

12.6 Findings

Finding	Severity	Description
SEC-04	MEDIUM	No token revocation; compromised JWTs valid until expiry. Logout endpoint missing.
SEC-05	MEDIUM	PII / PHI not filtered in prompts or responses; regulatory risk.

Performance Profile

13.1 Measured Latency

Operation	P50	P95	P99
Node identity lookup	45 ms	120 ms	200 ms
N-hop traversal (2-hop)	180 ms	450 ms	800 ms
N-hop traversal (3-hop, blocked)	—	—	—
Shortest path query	250 ms	600 ms	1.2 s
Fact extraction (10 facts)	90 ms	220 ms	400 ms
MCP tool round-trip overhead	+30 ms	+80 ms	+150 ms
Agent single-tool query	2.0 s	4.5 s	8.0 s
Agent multi-tool query (3 tools)	6.0 s	12.0 s	20.0 s
First token to UI	0.8 s	1.8 s	3.5 s

13.2 Throughput Envelope

Metric	Value
Concurrent users (local Docker)	5–10
Concurrent users (ECS production)	50–200
Neo4j Cypher queries / sec	200–500 (warm pagecache)
Agent requests / minute	20–60 (LLM-bound)

13.3 Bottlenecks

Three latency bottlenecks dominate. First, LLM inference, which accounts for the bulk of agent multi-tool query time and is essentially out of the platform's control. Second, deep-hop fan-out, which is mitigated by the two-hop cap but should be reinforced with graph indexes on hot label/property combinations. Third, MCP transport overhead, which is small per call but accumulates over multi-tool flows; the audit recommends evaluating direct in-process tool execution for the top three highest-frequency tools.

13.4 Findings

Performance findings appear under the database (DB-01, DB-02, DB-03) section above. There is no LLM cost instrumentation today; this is tracked as an operational risk (OR-04) and is recommended for early remediation.

Security Posture

14.1 Threat Model

Threat	Mitigation	Status
Cypher injection	Parameterised queries throughout	Strong
JWT tampering	HS256 signature; per-tier validation	Strong
MCP MITM	TLS — currently disabled (SEC-01)	GAP
Agent prompt injection	System-prompt scope, schema-validated tools	Adequate
Tool overreach (python_repl)	Tool always loaded (AGT-05)	GAP
DoS via concurrent requests	No rate limiting (SEC-03)	GAP
Container CVEs	No Trivy / ECR scan (recommended)	GAP
Dependency CVEs	No pip-audit (recommended)	GAP
JWT secret compromise	No rotation policy (TR-05)	GAP

14.2 PII / PHI

There is no PII or PHI filtering today, in either inbound prompts or outbound responses. The current data set is bounded to public sources and internal CSL deal data with no patient-level information; nevertheless, the regulatory environment for biomedical AI (FDA 21 CFR Part 11; HIPAA where applicable) suggests that an AWS Comprehend-based redaction middleware should be in place before any patient-derived data is introduced.

Code Quality and Engineering Discipline

15.1 Style and Linting

Black, Ruff, and isort are in use across the four service code bases. A pre-commit hook enforces formatting and import ordering before commit; the CI pipeline re-runs the same checks. Type hints are used throughout, with Pydantic v2 providing runtime validation at the API boundary.

15.2 Tests

Tests are co-located with the code they cover (graph_mapper.py / graph_mapper_test.py), with shared fixtures in per-package confest.py. Pytest-xdist parallelises execution. A Neo4j testcontainer is used for integration tests; coverage is reported to reports/coverage/lcov.info.

15.3 Evaluation

Engineering discipline is high. The codebase reads consistently; reviewer burden is moderate; tests are present at both unit and integration levels. The two engineering improvements worth prioritising are (a) pip-audit in CI to catch vulnerable dependencies in the 175-package requirements file, and (b) end-to-end agent eval in MLflow as a regression baseline before any LLM model upgrade.

Observability

16.1 What Is Monitored

- Liveness: each service exposes /health (returns 200 if the process is up).
- API canaries: 5 of the 20 Core API endpoints are polled by the canaries service; CloudWatch alarms fire on failure.
- Logs: ECS task stdout streams to CloudWatch in AWS; Docker logs locally.

16.2 What Is Not Monitored

- Structured logs — current logging is plain text, incompatible with CloudWatch Logs Insights queries.
- Metrics — there is no Prometheus or CloudWatch metrics exporter; throughput, error rate, and latency are not captured at the application layer.
- Distributed tracing — there is no OpenTelemetry / X-Ray integration; cross-service requests must be reconstructed by timestamp correlation.
- LLM cost — token consumption and cost per query are not captured; per-user / per-team chargeback is not currently possible.
- Deep health — /health does not check Neo4j, MCP, or LLM availability.

16.3 Findings

Finding	Severity	Description
OBS-01	HIGH	No structured JSON logging.
OBS-02	HIGH	No metrics export.
OBS-03	HIGH	No distributed tracing.
OBS-04	HIGH	Shallow /health endpoints.

Operational Runbook

17.1 Routine Operations

- Deploy a new image: build via bin/docker/ecr/build.sh, push via bin/docker/ecr/push.sh, then aws ecs update-service --force-new-deployment.
- Restart Neo4j after a heavy analytics query: documented in docs/maintenance.md; manual via ECS console.
- Refresh ALB self-signed certificate: see scripts under bin/alb/.

17.2 Manual Rotations

Two rotations are currently manual and deserve attention. First, the JWT signing secret has no rotation policy; rotation requires coordinated restart across all four service tiers. Second, the Neo4j SSL certificate expires near year-end and is renewed manually. Both should be lifted into automated workflows; both are listed in the documented risk register.

17.3 Common Issues

- Network hiccups between developer laptops and the difflabs AWS environment, observed roughly once or twice per month, typically resolving within 10 minutes.
- ALB target groups occasionally need a manual reset during deployments.
- Similarity endpoint requires a projected graph that is built out-of-band; this projection is not persisted across Neo4j restart.

PART V — DEPLOYMENT AND ENVIRONMENTS

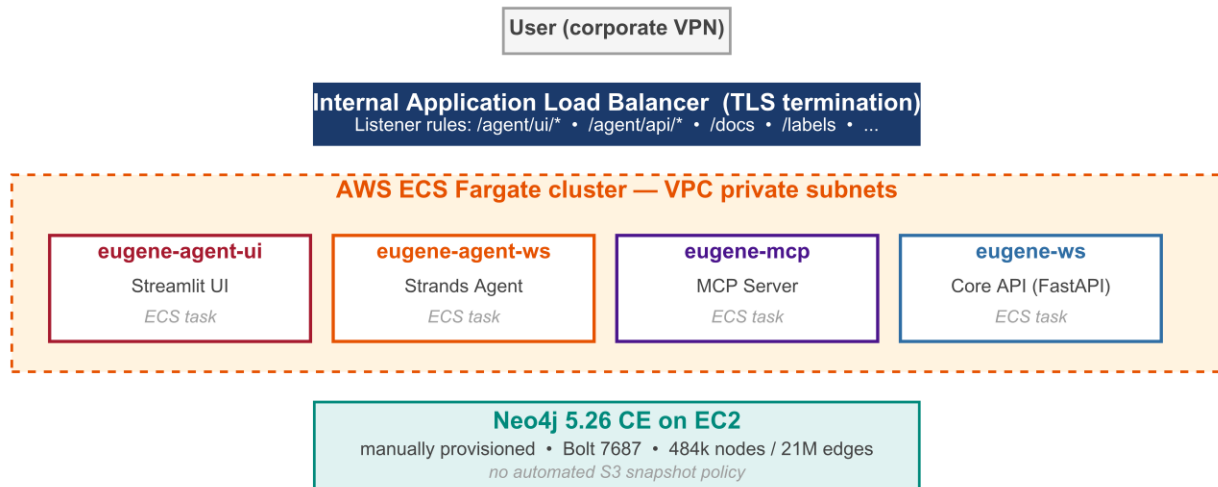
AWS Topology

18.1 Environment Matrix

Attribute	Local	Difflabs (Dev)	AIA (Production)
Platform	Docker Compose	AWS ECS Fargate	AWS ECS Fargate
Region	—	us-east-1	eu-central-1
Account	—	087084717211	010928221940
Auth	Local JWT bypass	Entra ID	Entra ID
Neo4j	Container	ECS task	EC2 (manual)
Secrets	docker.env	Secrets Manager	Secrets Manager
TLS	Disabled	ALB-terminated	ALB-terminated
IaC root	—	difflabs/iac/	infrastructure/

18.2 Production Topology

AWS Production Deployment Topology



Each environment runs an internal-only ECS Fargate cluster behind an internal Application Load Balancer with TLS terminated at the ALB. Container images live in ECR (separate registries per environment). The production Neo4j is provisioned manually on EC2; staging Neo4j is an ECS task. Listener rules route `/agent/api/*` to the agent service, `/agent/ui/*` to the Streamlit UI, `/docs` and other root paths to the Core API.

18.3 Terraform Module Structure

- `modules/eugene-services` — ECS service and task definitions.
- `modules/eugene-containers` — ECR repositories and image config.
- `modules/eugene-ec2` — EC2 baseline for cluster nodes.

- modules/eugene-ecs-database — Neo4j ECS task and storage.
- environments/ — environment-specific tfvars (qa, prod).

18.4 CI / CD

Code pushed to the repository triggers pre-commit hooks (Black, Ruff, isort), then GitHub Actions runs unit tests, integration tests against a Neo4j testcontainer, builds the Docker image, pushes to ECR, and runs a Terraform plan. Production deploys require a manual approval gate before Terraform apply triggers a rolling ECS update.

Local Development

19.1 Compose Layout

docker-compose.yml at the repository root defines five services that start in dependency order via depends_on: condition: service_healthy: neo4j → eugene_ws → eugene_mcp → eugene_agent_ws → eugene_agent_ui. A second UI container, eugene-agent-ui-next, runs the Next.js v2 frontend on port 18502.

19.2 Port Convention

Service	Local port	Container port
Neo4j Browser	17474	7474
Neo4j Bolt	17687	7687
Core API	18000	8000
MCP Server	18443	8000
Agent Backend	18001	8000
Streamlit UI	18501	8501
Next.js UI (v2)	18502	18502

19.3 docker.env Pattern

All configuration travels via docker.env (excluded from version control). Required keys include NEO4J_URI, NEO4J_USERNAME, NEO4J_PASSWORD, ENVIRONMENT (set to local for dev), EUGENE_CLIENT_SECRET, OPENAI_API_KEY and / or ANTHROPIC_API_KEY, LLM_PROVIDER, and EUGENE_MCP_SERVER_URL. The local stack can be pointed at the AWS-hosted Neo4j by overriding NEO4J_URI; corporate VPN required.

PART VI — RISK, CONCLUSION AND APPENDICES

Documented Risk Register

20.1 Technical Risks

ID	Risk	L	I	Status
TR-01	LLM API rate limit under concurrent load	Med	High	Active

ID	Risk	L	I	Status
TR-02	Neo4j deep N-hop performance degradation	Med	High	Active
TR-03	Prompt injection via user input	Low	High	Active
TR-04	Agent hallucination of biomedical data	Med	High	Active
TR-05	JWT secret compromise	Low	Critical	Documented
TR-06	Dependency vulnerability in 175-pkg requirements	Med	Med	Active
TR-07	Neo4j data loss on container restart	Low	Critical	Mitigated
TR-08	MCP server unavailability blocking all queries	Med	High	Planned

20.2 Operational Risks

ID	Risk	L	I	Status
OR-01	Knowledge graph data staleness	Med	High	Active
OR-02	LLM model version change breaking behaviour	Med	Med	Documented
OR-03	Entra ID tenant configuration drift	Low	High	Planned
OR-04	AWS cost overrun from LLM API usage	Med	Med	Active
OR-05	Single-region deployment (eu-central-1)	Low	High	Documented

20.3 Mitigation Priority

- Immediate action — TR-02 (deep query performance): graph indexes; the two-hop cap is already in place.
- Monitor and plan — TR-05 (JWT compromise) and TR-07 (Neo4j data loss): rotation policy, S3 snapshot policy.
- Planned improvements — TR-01, TR-04, TR-06 and OR-04: queueing, citation enforcement, pip-audit, cost instrumentation.

Programme-Level Conclusion

21.1 Where Eugene Stands

Eugene is a credible, well-architected first generation of CSL's biomedical agentic platform. The choice of microservice + DDD + hexagonal patterns has paid dividends in clarity and testability; the Strands ReAct agent provides flexible reasoning over a constrained tool surface; and the Neo4j-centric data model captures

relationships that flat search cannot.

21.2 Where the Gaps Lie

The platform is missing the operational and security spine that an enterprise rollout demands: deep observability, explicit timeouts, tested backups, RBAC on sensitive data, and a path away from monolithic agent reasoning toward a specialist-agent topology. None of these gaps are surprises — they are documented in the source materials and reproduced in this evaluation. They are a programme of work, not a re-architecture.

21.3 Recommendation

We recommend proceeding to the four-phase remediation and next-generation roadmap set out in the companion document, with the eight-week security and operational backbone as the first commitment. The platform's architectural strengths mean that hardening work compounds rather than displaces what already works. With those investments, Eugene transitions from a credible internal tool to a strategic CSL asset.

Appendix A — Glossary

Term	Meaning
ALB	Application Load Balancer (AWS)
APOC	Awesome Procedures On Cypher (Neo4j plugin library)
DDD	Domain-Driven Design
ECR	Elastic Container Registry (AWS)
ECS	Elastic Container Service (AWS)
GDS	Graph Data Science (Neo4j plugin)
HS256	HMAC SHA-256 (symmetric JWT signing)
IaC	Infrastructure as Code
MCP	Model Context Protocol (FastMCP implementation)
Milvus	Open-source vector database
NCT ID	ClinicalTrials.gov trial identifier
PMID	PubMed unique identifier
RBAC	Role-Based Access Control
ReAct	Reason–Act–Observe agent pattern
RS256	RSA SHA-256 (asymmetric JWT signing)
SSE	Server-Sent Events
UPN	User Principal Name (Entra ID identifier)
VPC	Virtual Private Cloud (AWS)

Appendix B — Core API Endpoint Inventory

Domain	Method	Path	Notes
Auth	GET/POST	/login, /auth/callback, /auth/whoami	Entra ID flow
System	GET	/health	Liveness only

Domain	Method	Path	Notes
System	GET	/release-notes	Static content
Stats	GET	/stats	Graph counters
Foundation	GET	/labels/{label}	Paginated
Foundation	GET	/count/{label}	
Foundation	GET	/node/find/{value}	Fuzzy match opt
Foundation	POST	/node/details	Up to 50 IDs
Foundation	GET	/graph/relationship/start/{id}	n_hop ≤ 2
Foundation	GET	/graph/facts/start/{id}	n_hop = 1
Foundation	GET	/graph/path/start/{id}/end/{id}	Shortest path
Foundation	GET	/graph/reachability/start/{id}/end/{id}	
Drug	GET	/drugs/aliases/{name}	
Drug	GET	/drugs/aliases/id/{drug_id}	
Patent	GET	/patents/drugs	
Patent	GET	/patents/clinicaltrials	
Patent	GET	/patents/geneproteins	
Patent	GET	/patents/count	
PubMed	GET	/pubmed/drugs	
PubMed	GET	/pubmed/clinicaltrials	
PubMed	GET	/pubmed/geneproteins	
PubMed	GET	/pubmed/count	
Org	GET	/organizations/{pattern}	Wildcards
Org	GET	/organizations/assets/{id}	
Search	POST	/similarity/{label}	Embedding
Search	POST	{label} (faceted)	Facet body

Appendix C — MCP Tool Inventory

#	Tool	API
1	fetch_identity	/auth/whoami, /node/find
2	fetch_by_label	/labels/{label}
3	fetch_similar	/similarity/{label}
4	lookup_node_by_value	/node/find/{value}
5	fetch_node_details	/node/details (POST)
6	fetch_drug_aliases	/drugs/aliases/{name}
7	fetch_facts	/graph/facts/start/{id}
8	fetch_node_relationships	/graph/relationship/start/{id}
9	fetch_paths	/graph/path/start/{id}/end/{id}
10	has_reachable_path	/graph/reachability/...

#	Tool	API
11	find_organization_names	/organizations/{pattern}
12	find_organization_assets	/organizations/assets/{id}

Appendix D — Source Documents Consulted

- EUGENE_PROJECT_VALIDATION_DOCUMENT.md (v1.0, 30 March 2026).
- EUGENE_STAGE1_ASSESSMENT_REPORT.pdf (April 2026); content extracted via generate_stage1_report.py source.
- DEVELOPER_GUIDE.md (current as of April 2026).
- docs/architecture/eugene-architecture.drawio (canonical architecture diagram).
- EUGENE_COMPLETE_DOCUMENTATION.md and EUGENE_COMPLETE_VALIDATION_DOCUMENT.md.
- ROUTE_DEEP_DIVE.md, oauth.md, entra_id.md, dns.md, maintenance.md, csl_aws.md, terraform.md, docker.md, local_dev_setup.md.