ALPHA Architecture

This document separates **what exists today** (a front-end reference implementation with mock data) from the
target architecture the UI is designed to operate.

---

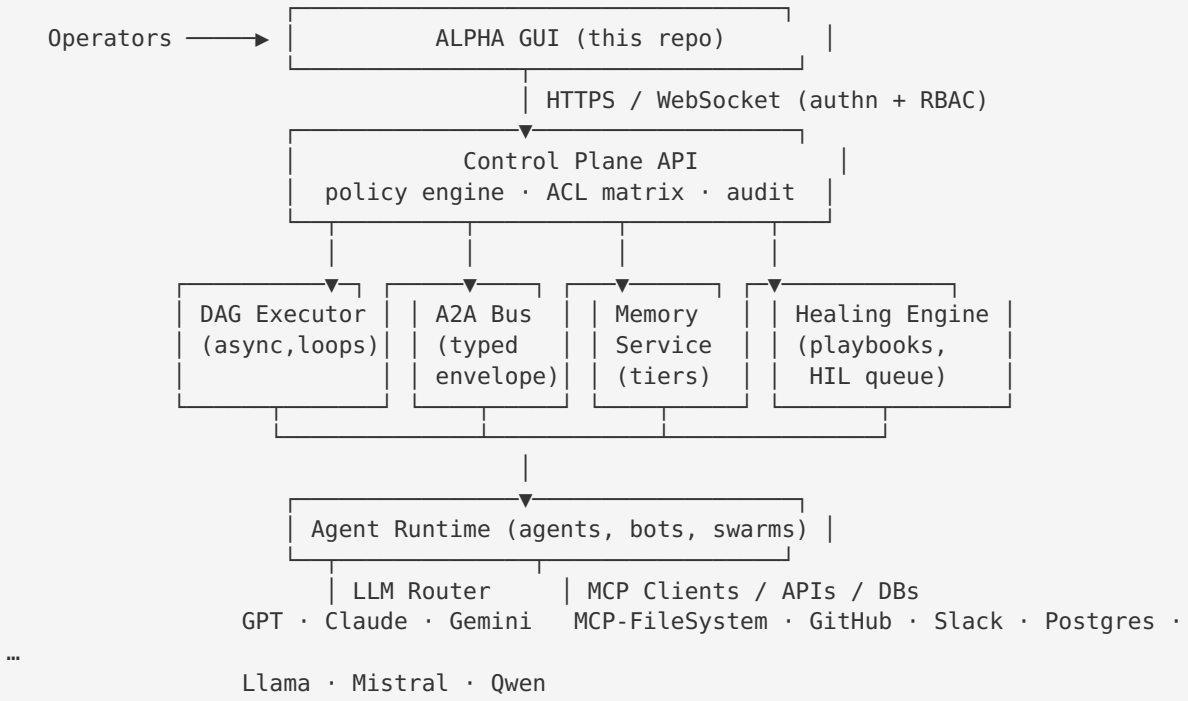
1. Current Architecture (implemented)

```
Browser
 └─ app/layout.tsx          (fonts, forced dark theme, metadata)
     └─ app/page.tsx        → <AlphaShell/>
         ├─ AlphaSidebar     (10 sections, collapsible)
         └─ active page      (next/dynamic, ssr:false, code-split)
               ├─ components/pages/*-page.tsx
               ├─ components/alpha-ui/*   (StatCard, HealthRing, MiniMesh, Activity-
Feed, PageWrapper)
               ├─ components/charts/*     (Recharts)
               └─ lib/mock-data.ts        (typed, deterministic mock data)
```

Concern	Implementation
Routing	Single route `/`; section switching is client state in `AlphaShell` (`NavSection` union in `lib/types.ts`)
Code splitting	Each page loaded with `next/dynamic` and `ssr: false`
State	Local React state per page; no global store, no persistence
Data	`lib/mock-data.ts` exports `agents`, `workflows`, `messages`, `swarms`, `mcpServers`, `memoryEntries`, `llmModels`, `healingEvents`, `activityFeed`, `dashboardStats`, `tokenUsageTimeSeries`, `healthMetrics`
Visualization	Custom SVG (mesh, mini-mesh, DAG canvas, health rings), Recharts (token/cost/latency)
Animation	Framer Motion + CSS keyframes
Backend	None. A Prisma schema scaffold exists but no models are used
Auth	None — the app opens directly to the Command Center
Security controls	None active; PQC / zero-trust labels are illustrative

---

2. Target Architecture (planned)



2.1 Asynchronous DAG Executor

- Workflows are versioned JSON graphs: `nodes[]`, `edges[]`, `triggers[]`, `policies`.
- Topological scheduling with async fan-out (Parallel Fork) and fan-in (Join); nodes run as soon as dependencies resolve.
- Per-node retry (exponential backoff + jitter), timeout, idempotency key, and compensation step.
- **Controlled Loop Engineering:** every Loop node declares `maxIterations`, `tokenBudget`, `timeoutMs`, and a convergence predicate; exceeding any bound halts the loop and raises a Self-Healing event.

2.2 A2A Message Envelope

```
{
  "id": "msg_…", "traceId": "…", "type": "task|result|query|broadcast|error",
  "sender": "agent:orch-prime", "recipients": ["agent:code-alpha"],
  "priority": "low|medium|high|critical", "ttlMs": 30000,
  "payload": { }, "signature": "<ML-DSA or Ed25519 over canonical envelope>"
}
```

The bus logs envelope metadata for audit but treats payloads as opaque.

2.3 Swarm Coordination Strategies

Strategy	Behavior
Round-robin	Distribute tasks evenly
Consensus	N agents answer; majority / judge selects
Competitive	Agents race; first valid result wins, others cancelled
Hierarchical	Lead agent decomposes and delegates
Emergent	Agents claim tasks from a shared board based on capability scores

2.4 Self-Healing Loop

`Detect → Diagnose → Heal → Verify → Resolve`. Automatic actions (retry, model fallback, reroute, memory flush,
agent restart) are allowed only for P2/P3 faults; P0/P1 and any irreversible action route to a human approval queue.

2.5 Security Layers (target)

1. Operator authn + RBAC at the GUI/API boundary
2. Per-agent workload identity, short-lived credentials
3. Deny-by-default agent × tool ACL matrix, enforced server-side
4. Hybrid PQC key exchange for inter-service channels (X25519 + ML-KEM, FIPS 203)
5. Signed, append-only audit log (ML-DSA FIPS 204 / SLH-DSA FIPS 205)
6. Secrets in a managed vault; never in prompts, memory, or logs

See docs/THREAT_MODEL.md.

---

3. Design System

Token	Value
Base	`#0A0A0F`
Neon cyan	`#00F5FF`
Electric purple	`#8B5CF6`
Hot pink	`#EC4899`
Surfaces	`glass-panel`, `glass-panel-hover`
Fonts	DM Sans · Plus Jakarta Sans · JetBrains Mono