

MIO — A Shared Memory and Knowledge Layer for AI Agents

Product Specification

Version 1.2 · September 2026

ArtminAI · Nik Nastev

1. What MIO Is

MIO is a **memory and knowledge layer** that sits underneath all of an organization's AI tools and gives them a shared, persistent context.

Today every AI tool remembers only itself. Claude doesn't know what you wrote in ChatGPT. Tomorrow's session doesn't know what yesterday's decided. One agent discovers a problem; another rediscovers it a week later.

MIO solves exactly this: **one source of truth that every agent reads from and writes to** — over a standard protocol (MCP), with no changes to the tools themselves.

In short: if AI agents are employees, MIO is their shared documentation, archive, and working memory.

2. The Problem It Solves

Without MIO	With MIO
Every chat starts from zero	The agent loads context in seconds
Knowledge lives in one person's head	Knowledge is written down, searchable, shared
Decisions get revisited three times	Decisions are recorded with reason and date
The same bug gets debugged twice	Bugs are stored with symptom and fix
Changing model or tool = losing everything	Memory is independent of the vendor
A new hire = months of onboarding	A new hire (or agent) asks the memory

3. Core Principle: No Vendor Lock-In

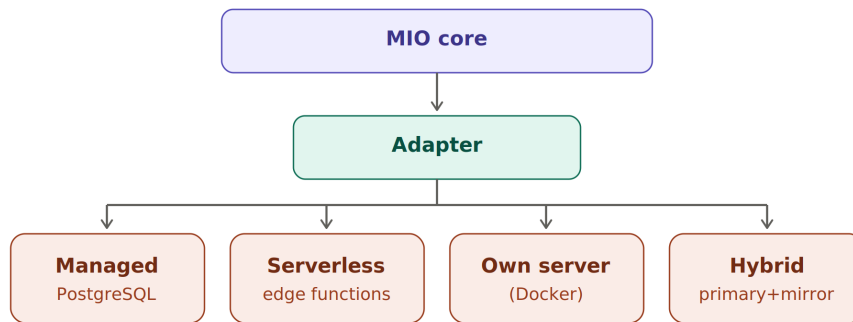
MIO is not a service on a particular platform. MIO is an architecture.

The product consists of three things, all of which are yours and stay yours:

1. **Protocol layer** — an MCP server with a catalog of tools. This is the interface agents see.
2. **Logic layer** — rules for writing, deduplication, hybrid search, the entity graph, observability.

3. **Adapter layer** — a thin translation to whatever storage sits underneath.

Only the third layer knows where the data physically lives. The first two neither know nor care.

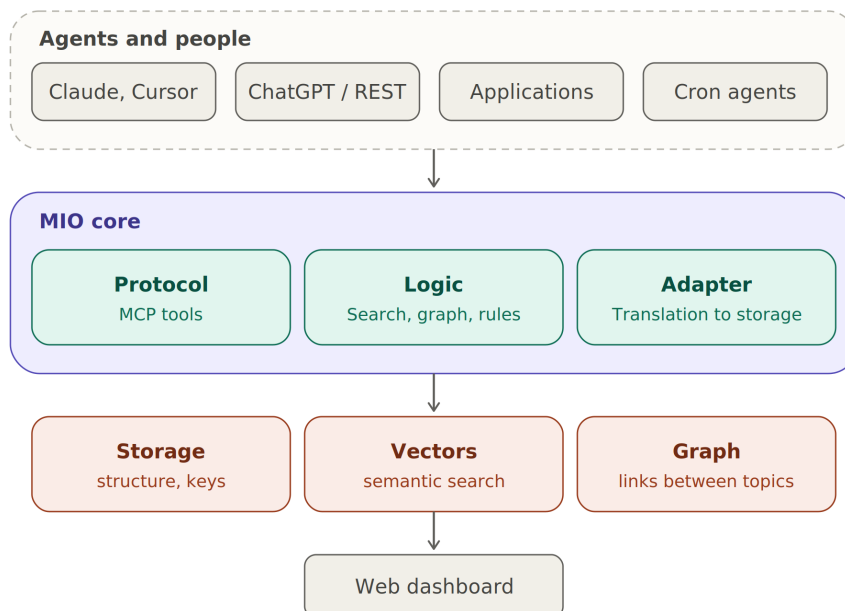


The same MIO — four different environments. Only the adapter changes.

What this means in practice:

- Today you run on a managed provider — tomorrow you migrate to your own server. The agents never notice.
- You can run in two places at once: primary storage plus a mirror at a different provider. On failure the system reads from the fallback automatically.
- No provider can raise your price or shut you down without you having an exit.
- Data lives in standard PostgreSQL — not in a proprietary format you can't get out of.

4. Architecture



MIO layers. The adapter is the only layer that knows where the data lives.

Principle: dual-write. Every record goes simultaneously into structured storage (exact, key-based lookups) and into the vector index (semantic search). The two paths are checked against each other.

Principle: dual-read with fallback. If the primary path fails, the system reads from the mirror automatically instead of erroring out. The user sees an answer, not a failure.

5. Capabilities

5.1 Memory

The core of the system.

Capability	Description
Write	Every thought, decision, fact, or lesson is stored with type, project, tags, and links
Types	fact, decision, bug, pattern, lesson, preference, note, session
Hybrid search	Vector + full-text search fused via RRF — finds by meaning and by exact term
Key lookup	Precise retrieval of a specific record without searching
Update	Records are corrected in place; history is preserved
Forget	Soft delete (tombstone) — reversible, with an audit trail
Deduplication	Semantic — the same fact doesn't enter twice under different wording
Session memory	Automatic summary of a working session plus extracted key facts
Multi-agent	Each agent has an identity; sees the shared layer and its own

5.2 Entity Graph

Memory answers not only *"what have I said about X"* but also *"what is connected to X"*.

- Automatic recognition of entities: projects, platforms, people, clients, tools, topics
- Deterministic extraction (dictionary + rules) — no model cost on every write
- Weighted links, sorted by strength
- Answers questions like: *"which projects use this tool"*, *"what depends on this database"*, *"what belongs to this client"*
- The graph is folded into search results automatically — it requires no new way of asking

Reference figure: on a real corpus of ~12,000 records the graph produces **11,859 links**, with a ceiling against dilution and coverage monitored weekly.

5.3 Project Memory

Every project gets its own dossier with four mandatory parts:

1. **Snapshot** — where the project stands right now: phase, progress
2. **Todo** — the single source for "what's left", each line with a verification criterion

3. **Decisions** — what was decided, when, and why
4. **Bugs and lessons** — what broke and how it was fixed

The agent starts a session with the actual state, not an assumption.

5.4 Knowledge Base and RAG

- **Document ingestion** — semantic chunking, embedding, indexing
- **Collections** — knowledge separated by topic (regulations, product documentation, internal policy)
- **Semantic search** across chunks, with optional model-based reranking
- **Categories and types** — personal / work / technical; note / document / publication
- **Automatic documentation sync** — external doc pages are monitored and re-indexed on change

5.5 Planner and Tasks

- Tasks with project, priority, status, and assignee
- Accessible to humans (dashboard) and agents (MCP) at the same time
- An agent can open, read, and close tasks as part of its work
- Task ↔ memory link: the decision that produced the task is one click away

5.6 Automatic Context

A single call at the start of a session returns what matters on the topic: decisions, bugs, patterns, tasks, knowledge. Token-optimized (~2,000) so it doesn't eat the context window.

Effect: **an agent that knows the history, without a human retelling it.**

5.7 Web Dashboard

For the people on the team who don't work through an agent:

- Browse and search the entire memory
- Notes — quick capture from phone or browser
- Projects and statuses
- Information / watch feed (optional)
- **Monitoring** — system health, capacity, spend, health indicators
- Served on the client's own domain

5.8 Observability and Health

Designed so that **silent failure is impossible.**

- Health signals for every critical path (embedding, write, mirror, quotas)
- Shows the **effective remaining budget**, not a misleading total balance
- Three states: healthy / limited / unconfirmed — "I don't know" is never presented as "all good"
- Scheduled checks for coverage, drift, and growth

- Alerts only on threshold — silence is part of the design, not absence of monitoring
- Full audit trail: which agent, when, what it wrote

5.9 Resilience and Portability

- **Bidirectional mirror** between primary and backup storage — at different providers, if you wish
- **Automatic fallback** when the primary path is unavailable
- **Drift audit** — a tool that compares both sides and backfills what's missing
- **Full backup** — schema, data, restore script, test procedure
- **Export at any time** — standard SQL dump plus vectors; nothing is locked in
- Secrets are stored separately from the data archive

6. MIO in an Organization with Many Employees

MIO's value grows with the number of people using it. For one person it's a convenience. For thirty, it's the difference between an organization that remembers and one that starts over every time.

6.1 Why Scale Works in the System's Favor

Knowledge in a team doesn't add up linearly — it intersects. Thirty people produce hundreds of overlapping questions: *what does this client want, why was this approach chosen, where is that document, what broke last time*. Today those answers are found by asking a colleague — the most expensive kind of search, because it stops two people at once.

MIO turns that into a query against memory. The more people write to it, the more often the next person finds a ready answer.

Headcount	What changes
1-5	Personal memory, continuity between sessions
5-20	Shared project context, end of repeated explanations
20-100	Organizational memory — onboarding drops from months to days
100+	Departments split into spaces; shared knowledge stays common

6.2 Spaces: Organizational, Team, Personal

Memory is divided into three levels:

- **Organizational** — policies, processes, product knowledge, regulations. Readable by everyone.
- **Team / project** — the project dossier: decisions, bugs, tasks, status. Readable by the team and by agents working on it.
- **Personal** — an individual's notes, drafts, working hypotheses. Never leaks into the shared layer unless the person publishes it.

The separation is enforced at the route and permission level, not by an agreement not to look. An employee physically cannot see someone else's personal space.

6.3 Roles and Permissions

Role	Can
Reader	Search and read the organizational and their own team space
Contributor	Plus write to their team and personal space
Project owner	Plus edit and close records within their project
Administrator	Permissions, spaces, keys, integrations, audit
Agent	Permissions inherited from the person or team it acts on behalf of

Critical: **an agent never has more permissions than the human who launched it.** If the employee can't see the finance space, neither can their agent.

Status: spaces, isolation, and scoped keys work today. The role-based permission model is designed and is the next development phase — at deployment it is finalized against the client's actual org structure. Stating this explicitly so nothing is bought that hasn't shipped.

6.4 Onboarding a New Employee

A new hire gets access to the organizational space and their team, and from day one asks questions of the memory instead of colleagues:

- *Why do we use this vendor and not the other one?*
- *Who owns client X and what was agreed?*
- *How do we run a production deploy here?*
- *What broke the last time we touched this?*

Answers come with date, author, and source. The mentor steps in where the memory is empty — and the gaps are visible, so they get filled.

6.5 Accountability and Audit

- Every record carries **author, agent, timestamp, and source** — it's known who contributed what
- Soft delete with a traceable trail — nothing disappears silently
- Coverage report: which projects have a current dossier and which have fallen behind
- Contribution report: which teams feed the memory and which only read — a management signal, not a punishment

6.6 What Grows with Headcount (and Must Be Planned)

Honest about cost, because at organizational scale it isn't negligible:

Grows	How it's managed
Record and vector count	Vectors are ~80% of volume; dimensionality can be reduced without re-embedding
Embedding spend	Capped key per organization or team, visible in the dashboard
Traffic and requests	Caching, batch operations, measurement per call rather than by intuition
Noise in memory	Semantic deduplication plus rules for what deserves a record
Permissions and spaces	Admin panel; at 100+ people, sync with the client's directory

Database size at ~12,000 records is around 400 MB. A disciplined 50-person organization reaches that volume in a few months — meaning tens of gigabytes a year, not terabytes.

6.7 The Condition You Can't Work Around

MIO is not a tool you install and it works. At organizational scale, success depends on one thing:

It must be clear when memory gets written — and that must be an obligation, not goodwill.

In practice this means a few rules embedded in the workflow: project ends = dossier written; decision made = reason recorded; bug fixed = symptom and fix recorded. Agents automate much of this, but the rule has to exist.

Organizations that adopt MIO without that rule end up with an empty database and disappointment. That's why phase 3 of the rollout is about working habits, not technology.

7. Integrations

Available today:

Channel	How
Claude.ai	MCP connector
Claude Code / CLI	MCP
Cursor, VS Code, Copilot CLI	MCP
ChatGPT / any model	REST API
Google Workspace (Gmail, Calendar, Drive)	connector
Microsoft 365	connector
Telegram / email	notifications
Voice interface	custom transport
Cron / scheduled agents	REST
Your own applications	REST + scoped key

Principle: MIO doesn't replace your tools. It plugs in underneath them.

8. Security

Measure	Implementation
Data isolation	Each client — own route prefix and own keys
Row Level Security	Enabled on all data tables
Public access	Explicitly revoked for memory and the entity graph
Scoped keys	Separate keys for read, write, and admin operations
Secrets	Separate store, never in archives or logs
Leak protection	Automated guards on every deploy
Audit	Every record carries author, agent, time, and source
Model spend	Embedding runs through a capped key — controlled and visible cost

9. Technical Requirements

MIO requires no specific platform. It requires **four capabilities**, available from many different providers:

Capability	What's needed	Example options
Structured storage	PostgreSQL 14+	Managed PostgreSQL, self-hosted, any cloud
Vector index	pgvector with HNSW	The same database, or a separate vector store
Code execution	Serverless functions or a Node process	Edge functions, container, VPS
Scheduled jobs	Cron	pg_cron, system cron, provider scheduler

Core: TypeScript / Node.js, functional style, no dependency on any single cloud's SDK.

Embedding: a model of the `text-embedding-3-small` class (Matryoshka — dimensionality can be truncated without re-embedding). The provider is a configuration change; local models are supported.

Dashboard: static frontend — runs from any CDN, object storage, or your own nginx.

Proven Environments

The core runs in production on three different combinations:

- Managed PostgreSQL + edge functions + CDN hosting
- A serverless platform with integrated database and vector search
- Local Docker (PostgreSQL + Node) on an owned machine

Migrating between them is a configuration change plus a data transfer — not a rewrite.

Figures from a Real Deployment

Metric	Value
Memory records	~12,300
Vectorized	100% of the normal path
Graph links	11,859
Search latency (cloud database)	100-160 ms
Search latency (local instance)	~1 ms
Full context at session start	~2-4 s
Key uniqueness	12,126 / 12,126, zero duplicates
Database size at 12k records	~400 MB (80% of it vectors)

10. Deployment Models

A. Managed Instance

Hosted and maintained by us, on infrastructure of our choosing. The client gets an isolated environment, keys, and dashboard.

For: small teams, fast start, no internal DevOps.

B. Your Infrastructure, Our Support

We install MIO on the client's own account and provider — whichever it is. Bills and data belong to the client; we maintain the layer.

For: companies with an established cloud practice.

C. On-Premise

The entire layer runs in Docker inside the client's network. Data never leaves it.

For: regulated sectors, sensitive data.

Note: embedding can stay external or move to a local model — your choice.

D. Hybrid

Primary storage in one place, mirror in another. Automatic fallback if one fails.

For: critical workloads and teams that don't want a single point of failure.

Moving A → B → C is designed in from the start. It isn't a migration project — it's configuration plus a data transfer.

11. How a Rollout Works

Phase 1 — Foundation

Environment selection, instance, keys, MCP connection to the client's tools, dashboard. First records.

Phase 2 — Knowledge transfer

Ingest existing documentation, project dossiers, internal policy. Configure collections and categories.

Phase 3 — Working habits

Rules for when the agent reads and when it writes. Session-record templates. Team training — memory only works if people write to it.

Phase 4 — Automation

Scheduled briefings, automatic documentation sync, alerts, entity graph.

Phase 5 — Monitoring and tuning (*continuous*)

Health signals, coverage, thresholds, cost optimization.

12. What MIO Is Not

Stated honestly, so there are no surprises:

- **Not a model.** MIO doesn't think or generate. It remembers and supplies. The model remains your choice.
- **Not a replacement for your application database.** It's memory for agents, not a transactional system.
- **Not self-running.** If agents don't write to it, it stays empty. Write discipline is part of the rollout, not an option.
- **Semantic search is not exact search.** For guaranteed-exact lookups, use a key or a structured filter.
- **Graph quality depends on the domain vocabulary.** Initial tuning is a required part of the work.
- **Portability isn't free.** Changing environments is planned for and documented, but it needs a scheduled window and reissued keys.

13. Roadmap

Priority	What
Next	Tag filtering in RAG search · vector dimensionality optimization · direct knowledge ingestion without an intermediate layer
After that	Model-based discovery of uncovered entities (graph phase 2) · declared links between records · extended role-based permissions
Later	Local embedding model · self-service instance creation · public API for partner integrations

14. Next Step

The fastest way to judge MIO is a **demonstration on your own data**: we load one real project or a set of documentation into a test instance and show the difference — the same question asked of an agent with and without MIO.

You choose the environment for the demo. If you have no preference, we run it on ours and move it afterwards.

Get in touch and we will agree on the scope.

Contact

Nik Nastev · ArtminAI
Varna, Bulgaria