Product Requirements Document (PRD): Multi-Agent AI Platform Backend

1. Purpose

To develop a next-generation, self-hosted, multi-agent AI platform that enables organizations to orchestrate teams of horizontal (general) and vertical (specialist) agents across multiple projects, empowering users to automate, collaborate, and accelerate knowledge work through a secure, extensible interface.

2. Product Summary

- **Product Name:** Multi-Agent AI Orchestration Platform
- **Stakeholders:** CISO, AI/Tech/Engineering teams, Business Analysts, End-Users
- **Target Users:** Knowledge workers, engineering teams, analysts, project managers, operations, and AI practitioners
- **Supported Environments:** Linux (Ubuntu), macOS
- **Deployment:** Self-hosted, single-machine at first, designed for future distributed/scalable deployments
- **Frontend:** Modern Next.js web client (lightweight, API-driven)
- **Backend:** Python FastAPI, PostgreSQL, Redis, vector DB, PyTorch (GPU)

3. Goals & Objectives

- **Multi-agent support:** Orchestrate both generalist and specialist AI agents with project/team contexts
- **Self-contained and open:** All components run on-premises with no cloud/SaaS requirements
- **Interoperability:** Expose all backend features via FastAPI APIs for use by any client (web, CLI, bots)
- **Configurability:** Admins can add/remove/configure agents, tools, and projects
- **Universal agent communication:** Any agent can communicate or delegate work to any other agent using a standardized message format
- **Memory and context:** Each project/team has persistent, retrievable memory for long-term context (vector DB, semantic search)
- **Scheduling:** Allow scheduling of recurring tasks and agent workflows (cron, intervals)
- **Extensible tools:** Tools can be easily registered, assigned to agents, and executed (supporting standards like MCP)
- **Observability & automation:** Built-in monitoring, logging, and self-healing/auto-fix agent capabilities
- **Security & governance:** RBAC, multi-tenancy, superadmin/project admin/user roles, full audit trail

4. Features & Requirements

4.1. Core Features

4.1.1. Project & Team Management

- Create/manage projects
- Assign users (admins, members) to projects
- Add/remove/configure agents per project
- Multi-tenancy: data and teams are isolated per project

4.1.2. Agent Management

- Register agent types (generalist or specialist) with custom system prompts and tool access
- Assign agents to project teams; configure tool access and models per agent
- Agents must have well-defined roles, memory access, and permitted toolsets
- All agent activity auditable (by project and system admins)

4.1.3. Tooling Integration

- Add/edit/remove tools in a global registry
- Assign tools per agent (tools may be external APIs, local functions, or MCP servers)
- Admin UI for tool/agent configuration

4.1.4. Agent-to-Agent Communication

- All communication over a message bus (initially Redis, abstracted for future distributed support)
- Message schema must be universal: sender, receiver, project ID, type, payload, timestamp
- Agents can send, receive, and reply to messages; direct, broadcast, or topic-based

4.1.5. Memory and Retrieval-Augmented Generation

- Store all relevant knowledge, artifacts, and conversation in project-specific long-term memory (vector DB)
- All agent and user queries can trigger semantic search for relevant prior memory
- Expose APIs for storing, updating, retrieving, and deleting memory objects

4.1.6. Scheduling & Automation

- Support one-off, recurring, and interval-based jobs (e.g., daily reporting, periodic data fetches)
- Scheduled tasks can invoke agents, run workflows, or perform admin/maintenance
- Schedules must be editable via UI/API

4.1.7. LLM Model Management

- Allow configuration/use of both local (GPU-accelerated) and remote LLMs
- Enable per-agent model configuration
- Model loading, batching, and resource management abstracted for future scalability

4.1.8. Monitoring & Self-Healing

- Structured logging for all major actions (API, agent, scheduler, memory, errors)
- Healthcheck endpoints for platform and models
- Monitoring agents/services can detect, alert, and attempt automated remediation of failures (with escalation to superadmin if necessary)

4.1.9. Security, RBAC & Audit

- Full JWT-based authentication
- Role-based access control (Superadmin, Project Admin, Project User)
- BFF (backend for frontend) pattern for safe UI integration
- API/user/agent action audit logs
- Multi-tenancy enforced at all API/data layers

4.2. Non-Functional Requirements

- **Performance:** Agents and APIs must respond within 2 seconds under typical load; long-running workflows should be asynchronous
- **Reliability:** Platform must not lose tasks or memory in the event of process restarts; must log errors and surface health issues
- **Portability:** All services must be installable locally (Ubuntu/macOS); Docker configuration provided for all components
- **Extensibility:** New agents, tools, and integrations must be pluggable via config/code-first workflow
- **Usability:** Web UI must require minimal training for basic workflows (project creation, team assignment, agent querying)
- **Observability:** Platform must surface metrics and logs for both system admins and project admins

---

5. User Stories

5.1. As a Superadmin

- I want to see all projects, users, agents, and system health in one place
- I want to add/remove tools, agent types, and manage LLM models centrally
- I want to receive alerts if an agent or process fails or misbehaves

5.2. As a Project Admin

- I want to create a new project and invite colleagues as users or co-admins
- I want to assemble a custom team of agents for my project (mixing generalists and specialists)
- I want to configure agent prompts, tool access, and model use for each agent
- I want to schedule periodic tasks for my team's agents
- I want to review logs, outputs, and audit activity in my project

5.3. As a Project User

- I want to interact with project agents (ask questions, submit tasks) and get timely responses
- I want to view past project outputs and knowledge
- I want to be notified of results or issues

---

6. Success Criteria & Metrics

- [ ] Platform can be deployed and run locally (PostgreSQL, Redis, vector DB, FastAPI, Next.js, GPU inference)
- [ ] Superadmins can manage projects, users, agents, and tools
- [ ] Agents communicate via message bus; any agent can request help from any other
- [ ] Project memory persists and is queryable (semantic search) per project
- [ ] Users can schedule and trigger automated workflows
- [ ] All major actions are logged and audit trails are available
- [ ] Role-based access and multi-tenancy are enforced at all times
- [ ] System alerts for health issues, failures, or errors
- [ ] Extensibility proven by adding a new agent type and tool with no downtime

---

7. Out of Scope (Initial Release)

- Distributed (multi-machine) deployment (will be designed for, but not implemented initially)
- Managed/cloud/SaaS integrations (local only at launch)
- Third-party login or SSO
- Mobile app clients (Web, CLI, and Telegram/Bot API only)
- Agent code generation/modification through UI (code-first config only at launch)

---

8. Assumptions & Risks

- All users have access to Ubuntu or macOS hardware for initial deployment
- Projects/teams will not exceed single-machine performance limits in first release
- Admins will manage all tool and agent registration centrally (no end-user scripting/plugins in v1)
- Hardware GPU support is available if using large models
- If open source LLM/embedding models prove insufficient for certain use cases, API fallback may be considered in future

---

9. Open Questions

- Should agent memory be optionally encrypted at rest for certain projects?
- How granular does tool permissioning need to be (per user, per agent, per project)?

- What are the escalation/auto-remediation rules for the monitoring agent (e.g., when is human intervention required)?
- Any regulatory/compliance requirements for log retention or access control?
- Minimum browser/OS versions to be supported for UI?

---

10. Appendices

- **API Spec:** See TDD.md for API contracts and model details
- **User Flows:** (To be detailed as wireframes/UI mockups)
- **Glossary:**
    - *Agent:* Autonomous LLM-driven entity with specific role/tools
    - *Tool:* API, library, or external service callable by an agent
    - *Project:* Isolated workspace with its own agents, memory, users
    - *Memory:* Long-term, vector-searchable semantic storage for agent context
    - *MCP:* Model-Context Protocol, emerging standard for agent-tool interop

---