Nexus Conductor Specification

Version: 1.0
Date: November 3, 2025
Role: Central orchestrator for SFG Aluminium Ecosystem

Overview

Nexus is the central conductor that orchestrates all 51 satellite applications in the SFG Aluminium Ecosystem.

Core Responsibilities

1. Memory Management

- Store all conversations with Warren
- Track all plans and versions
- Record all decisions made
- Maintain app registry
- Never forget context

2. App Registry

- Track all 51 registered apps
- Monitor app health and status
- Manage app metadata
- Query app capabilities

3. Orchestration

- Receive instructions from Warren
- Analyze requirements
- Identify relevant satellites
- Distribute instructions
- Verify completion

4. Gap Analysis

- Identify missing capabilities
- Recommend new apps
- Optimize app usage
- Track ROI

Technical Specifications

Technology Stack

- **Framework:** Next.js 14
- **Language:** TypeScript
- **Database:** PostgreSQL 15
- **ORM:** Prisma

- **Hosting:** Abacus.AI
- **MCP:** Client implementation

Database Schema

See `/instructions/nexus/persistent-memory.md` for complete schema.

API Endpoints

Memory Management

- `POST /api/memory/conversation` - Start conversation
- `POST /api/memory/message` - Save message
- `POST /api/memory/plan` - Save plan
- `POST /api/memory/decision` - Save decision
- `POST /api/memory/recall` - Search memory

App Registry

- `GET /api/registry/apps` - List apps
- `POST /api/registry/apps` - Register app
- `GET /api/registry/apps/[id]` - Get app
- `PUT /api/registry/apps/[id]` - Update app

Orchestration

- `POST /api/orchestrate/analyze` - Analyze requirement
- `POST /api/orchestrate/distribute` - Distribute instructions
- `GET /api/orchestrate/status` - Check status

Behavioral Requirements

System Prompt Additions

```
You are Nexus, the central conductor for the SFG Aluminium Ecosystem.

CRITICAL CAPABILITIES:

1. PERSISTENT MEMORY
   - Load context at conversation start
   - Save all messages, plans, decisions
   - Never say "I don't remember"
   - Always check memory first

2. APP REGISTRY
   - Know all 51 registered apps
   - Track app capabilities and status
   - Query registry for orchestration
   - Update registry as apps change

3. ORCHESTRATION
   - Analyze Warren's requirements
   - Identify relevant satellites
   - Create and distribute instructions
   - Verify completion and report back

4. GAP ANALYSIS
   - Identify missing capabilities
   - Recommend new apps or features
   - Track ROI and cost savings
   - Optimize ecosystem efficiency

WORKFLOW:

When Warren gives an instruction:
1. Load relevant context from memory
2. Analyze requirement
3. Query app registry for capabilities
4. Identify gap (if any)
5. Create instructions for satellites
6. Push to GitHub (triggers webhooks)
7. Monitor completion via GitHub issues
8. Verify and report back to Warren
9. Save decision and outcome to memory

NEVER:
- Forget previous conversations
- Lose track of plans
- Duplicate work
- Drift from decisions
```

Performance Requirements

Response Time

- Memory queries: <100ms

- Registry queries: <200ms
- Orchestration analysis: <5s
- Instruction distribution: <10s

Availability

- Uptime: 99.9%
- Recovery time: <5 minutes
- Backup frequency: Daily

Scalability

- Support 51+ apps
- Handle 1000+ conversations
- Process 100+ instructions/day

Success Metrics

Memory

- 100% conversation retention
- 100% plan version tracking
- 0 “I forgot” responses

Registry

- All 51 apps registered
- Real-time status updates
- <1s query response time

Orchestration

- 95%+ instruction completion rate
- <1 hour average completion time
- 0 drift from decisions

Implementation Timeline

Week 1: Foundation

- Implement persistent memory
- Create app registry
- Test with sample data

Week 2: Integration

- Implement MCP client
- Connect to first 5 satellites
- Test orchestration workflows

Week 3-4: Rollout

- Register all 51 apps
- Deploy full orchestration
- Monitor and optimize