Persistent Memory System Guide

Overview

The persistent memory system allows NEXUS to remember conversations, plans, decisions, and application registrations across sessions, solving the critical "forgetting" problem.

Database Models

1. Conversation

Tracks all conversation sessions with metadata, status, and relationships.

Fields:
- `id` - Unique identifier
- `title` - Optional conversation title
- `startedAt` - When conversation began
- `lastActivityAt` - Last message timestamp
- `status` - ACTIVE, COMPLETED, ARCHIVED, SUSPENDED
- `userId` - Optional user identifier
- `metadata` - Additional context (JSON)

Relationships:
- Has many Messages
- Has many Plans
- Has many Decisions

2. Message

Stores all messages in conversations with role and content.

Fields:
- `id` - Unique identifier
- `conversationId` - Link to conversation
- `role` - USER, ASSISTANT, SYSTEM
- `content` - Message text
- `timestamp` - When message was created
- `metadata` - Tokens, model used, attachments (JSON)

3. Plan

Tracks implementation plans and their progress.

Fields:
- `id` - Unique identifier
- `conversationId` - Optional link to conversation
- `title` - Plan title
- `description` - Detailed description
- `status` - PENDING, IN_PROGRESS, COMPLETED, CANCELLED, ON_HOLD
- `priority` - LOW, MEDIUM, HIGH, CRITICAL

- `createdAt`, `updatedAt`, `completedAt` - Timestamps
- `metadata` - Task list, milestones, dependencies (JSON)

4. Decision

Records key decisions made during conversations.

Fields:
- `id` - Unique identifier
- `conversationId` - Optional link to conversation
- `planId` - Optional link to related plan
- `title` - Decision title
- `description` - Decision details
- `rationale` - Why this decision was made
- `madeAt` - When decision was made
- `madeBy` - Who made the decision
- `impact` - LOW, MEDIUM, HIGH, CRITICAL
- `metadata` - Alternatives, affected systems (JSON)

5. AppRegistry

Central registry of all satellite apps in the SFG ecosystem.

Fields:
- `id` - Unique identifier
- `appName` - Unique application name
- `appType` - CORE_SYSTEM, SATELLITE_APP, INTEGRATION, etc.
- `description` - App description
- `baseUrl` - Application URL
- `status` - ACTIVE, DEVELOPMENT, MAINTENANCE, DEPRECATED, ARCHIVED
- `technologies` - Tech stack (JSON array)
- `owner` - Responsible team/person
- `registeredAt`, `lastUpdatedAt` - Timestamps
- `repositoryPath` - Path to code repository
- `apiEndpoints` - Available endpoints (JSON)
- `metadata` - Dependencies, integrations, version (JSON)

6. Instruction

Stores reusable instructions and procedures.

Fields:
- `id` - Unique identifier
- `title` - Instruction title
- `content` - Full instruction text
- `category` - DEPLOYMENT, CONFIGURATION, TROUBLESHOOTING, etc.
- `priority` - LOW, MEDIUM, HIGH
- `usageCount` - How many times used
- `metadata` - Tags, related apps, prerequisites (JSON)

7. Context

Stores contextual information as key-value pairs.

Fields:
- `id` - Unique identifier
- `key` - Unique context key
- `value` - Context value (text)
- `category` - SYSTEM_CONFIG, USER_PREFERENCE, ENVIRONMENT, etc.
- `expiresAt` - Optional expiration for temporary context
- `metadata` - Scope, source, reliability (JSON)

8. KnowledgeBase

Builds organizational knowledge over time.

Fields:
- `id` - Unique identifier
- `topic` - Knowledge topic
- `content` - Full knowledge content
- `source` - Where knowledge came from
- `category` - TECHNICAL, BUSINESS_PROCESS, COMPLIANCE, etc.
- `tags` - Searchable tags (array)
- `relevanceScore` - For ranking (float)
- `metadata` - Related entities, confidence (JSON)

API Endpoints

Conversations API

Base: `/api/memory/conversations`

- `GET /api/memory/conversations` - List all conversations
- Query params: `page` , `per_page` , `status` , `userId`
- `POST /api/memory/conversations` - Create new conversation
- Body: `{ title?, userId?, metadata? }`
- `PATCH /api/memory/conversations/:id` - Update conversation
- Body: `{ title?, status?, metadata? }`
- `DELETE /api/memory/conversations/:id` - Delete conversation

Messages API

Base: `/api/memory/messages`

- `GET /api/memory/messages` - List messages
- Query params: `conversationId` , `page` , `per_page`
- `POST /api/memory/messages` - Create message
- Body: `{ conversationId, role, content, metadata? }`

Plans API

Base: `/api/memory/plans`

- `GET /api/memory/plans` - List plans
- Query params: `conversationId` , `status` , `priority` , `page` , `per_page`
- `POST /api/memory/plans` - Create plan
- Body: `{ conversationId?, title, description, priority?, metadata? }`

- `PATCH /api/memory/plans/:id` - Update plan
- Body: `{ title?, description?, status?, priority?, metadata? }`

Decisions API

Base: `/api/memory/decisions`

- `GET /api/memory/decisions` - List decisions
- Query params: `conversationId`, `planId`, `impact`, `page`, `per_page`
- `POST /api/memory/decisions` - Create decision
- Body: `{ conversationId?, planId?, title, description, rationale?, madeBy?, impact?, metadata? }`

App Registry API

Base: `/api/memory/app-registry`

- `GET /api/memory/app-registry` - List all apps
- Query params: `appType`, `status`, `page`, `per_page`
- `POST /api/memory/app-registry` - Register new app
- Body: `{ appName, appType, description?, baseUrl?, technologies?, owner?, repositoryPath?, apiEndpoints?, metadata? }`
- `GET /api/memory/app-registry/:id` - Get app details
- `PATCH /api/memory/app-registry/:id` - Update app
- Body: `{ description?, baseUrl?, status?, technologies?, apiEndpoints?, metadata? }`
- `DELETE /api/memory/app-registry/:id` - Delete app

Usage Examples

Creating a Conversation

```
const response = await fetch('/api/memory/conversations', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    title: 'Week 1 Implementation',
    userId: 'warren',
    metadata: { project: 'SFG Orchestration' }
  })
});
const { conversation } = await response.json();
```

Adding Messages

```
const response = await fetch('/api/memory/messages', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    conversationId: conversation.id,
    role: 'USER',
    content: 'Create the GitHub App for satellite apps'
  })
});
```

Registering an App

```
const response = await fetch('/api/memory/app-registry', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    appName: 'chronoshift-pro',
    appType: 'SATELLITE_APP',
    description: 'Advanced scheduling and payroll system',
    baseUrl: 'https://chronoshift-pro.abacusai.app',
    technologies: ['Next.js', 'TypeScript', 'PostgreSQL'],
    owner: 'Warren Heathcote',
    repositoryPath: 'apps/chronoshift-pro',
    metadata: {
      category: 'scheduling',
      staff_replaced: 2,
      annual_savings_gbp: 80000
    }
  })
});
```

Best Practices

1. **Always create a conversation** for significant interactions
2. **Record all important messages** with appropriate roles
3. **Track plans** for multi-step implementations
4. **Document decisions** with clear rationale
5. **Register all apps** as soon as they're deployed
6. **Use metadata** for rich context and searchability
7. **Update status** fields as work progresses
8. **Clean up old conversations** by archiving, not deleting

Troubleshooting

"Cannot find conversation"

- Ensure the conversationId exists in the database
- Check if conversation was accidentally deleted

"Duplicate app name"

- App names must be unique in the registry
- Check existing apps with GET /api/memory/app-registry

"Too many results"

- Use pagination with `page` and `per_page` parameters
- Default page size is 20 items

Implementation Status: Fully Operational
Location: `/home/ubuntu/sfg-nexus-mockup/app/prisma/schema.prisma`
API Location: `/home/ubuntu/sfg-nexus-mockup/app/app/api/memory/`