Workspace Collaboration Feature

A comprehensive Google Docs-like sharing system for Reflective Memory Kernel workspaces, enabling teams to collaborate on shared AI memory spaces.

Table of Contents

Introduction

The Workspace Collaboration feature transforms the Reflective Memory Kernel from a personal AI memory system into a collaborative platform. Just like Google Docs allows multiple users to work on the same document, this feature allows multiple users to share and interact with the same AI memory space (workspace).

What is a Workspace?

A **workspace** (internally called a "Group") is an isolated memory namespace where:

- AI conversations are stored and indexed
- Extracted entities of facts are saved
- Users can query and recall information
- All data is isolated from other workspaces

Why Collaboration?

In team environments, shared context is crucial:

- **Customer Support Teams**: Share customer interaction history across agents
- **Research Teams**: Collaborate on AI-assisted research with shared memory
- **Enterprise Knowledge**: Build organizational knowledge bases accessible to multiple employees

---

Problem Statement

Current Limitations

Before this feature, the existing group system had several limitations:

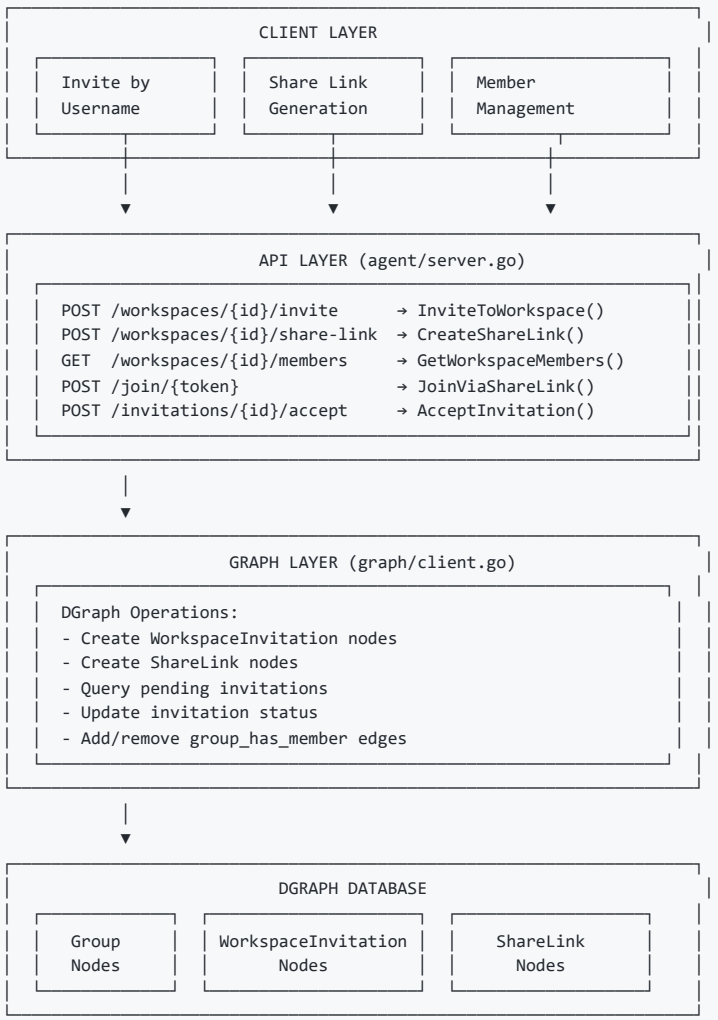
Limitation	Impact
No invitation workflow	Admins had to manually create subusers with passwords
No shareable links	Impossible to quickly share access to a workspace
Binary roles only	No granular control (e.g., read-only access)
No pending invitations	Users couldn't be invited before they registered

Goals

1. **Easy Sharing**: Generate shareable links like Google Docs
2. **Invitation System**: Invite existing users by username
3. **Admin Control**: Admins maintain full control over who can access
4. **Security First**: All access requires authentication
5. **Backward Compatible**: Existing groups continue to work

Solution Architecture

High-Level Architecture



Component Interaction

1. **Client** makes API request (e.g., invite a user)
2. **API Layer** validates JWT, checks admin permissions
3. **Graph Layer** creates/updates DGraph nodes
4. **DGraph** persists the data with proper indexes

Design Decisions

These decisions were made based on project requirements:

1. Authenticated Share Links

Decision: Users MUST be logged in to join via share link

Rationale:

- Prevents anonymous access to AI memory
- Maintains audit trail of who accessed what
- Aligns with enterprise security requirements
- Links are still “easy to share” - user just needs to login first

How it works:

```
User receives link → Clicks link → Redirected to login →
After login, automatically joins workspace → Access granted
```

2. Username-Only Invitations

Decision: Invitations are sent by username, not email

Rationale:

- No email infrastructure needed
- Simpler implementation
- Users must already have accounts
- Reduces complexity of pending invite management

Future Enhancement: Email invites could be added later with pending invite queue.

3. Admin/Subuser Role Model

Decision: Keep the existing binary role model (admin vs subuser)

Rationale:

- Simpler mental model for users
- Matches existing implementation
- Sufficient for most use cases
- Can be extended later if needed

Roles explained:

- **Admin**: Can do everything (invite, remove, delete workspace)
- **Subuser**: Can read/write memories, but cannot manage the workspace

4. Backward Compatibility

Decision: Existing groups continue to work without migration

Rationale:

- No data migration required
- Existing `group_has_admin` and `group_has_member` edges are reused
- New features are additive, not destructive

Role-Based Access Control

Permission Matrix

Action	Admin	Subuser	Non-Member
Read memories	✅	✅	❌
Write memories	✅	✅	❌
Invite users	✅	❌	❌
Create share links	✅	❌	❌
Remove members	✅	❌	❌
Delete workspace	✅	❌	❌
Spawn subusers	✅	❌	❌
View members list	✅	✅	❌
Leave workspace	❌*	✅	-

*Admins cannot leave - they must transfer ownership or delete the workspace.

How Permissions Are Checked

```
// Example: Before allowing an invitation
isAdmin, err := graphClient.IsGroupAdmin(ctx, workspaceNS, userID)
if !isAdmin {
    return http.StatusForbidden, "Only admins can invite users"
}
```

Data Models

WorkspaceInvitation

Represents a pending invitation for a user to join a workspace.

```
type WorkspaceInvitation struct {
    // DGraph unique identifier
    UID string `json:"uid,omitempty"`

    // DGraph type array (always ["WorkspaceInvitation"])
    DType []string `json:"dgraph.type,omitempty"`

    // The workspace (group) namespace this invitation is for
    // Example: "group_550e8400-e29b-41d4-a716-446655440000"
    WorkspaceID string `json:"workspace_id,omitempty"`

    // Username of the person being invited
    // Must be an existing registered user
    InviteeUserID string `json:"invitee_user_id,omitempty"`

    // Role they will receive upon accepting
    // Values: "admin" or "subuser"
    Role string `json:"role,omitempty"`

    // Current status of the invitation
    // Values: "pending", "accepted", "declined"
    Status string `json:"status,omitempty"`

    // When the invitation was created
    CreatedAt time.Time `json:"created_at,omitempty"`

    // Username of the admin who sent the invitation
    CreatedBy string `json:"created_by,omitempty"`
}
```

Lifecycle:

```
Created (status: "pending")
    |
    ├─→ User accepts → status: "accepted" → User added to workspace
    |
    └─→ User declines → status: "declined" → No action
```

ShareLink

Represents a shareable URL that allows authenticated users to join a workspace.

```
type ShareLink struct {
    // DGraph unique identifier
    UID string `json:"uid,omitempty"`

    // DGraph type array
    DType []string `json:"dgraph.type,omitempty"`

    // The workspace this link grants access to
    WorkspaceID string `json:"workspace_id,omitempty"`

    // Cryptographically secure random token (32 bytes, base64url encoded)
    // This is what appears in the URL: /join/{token}
    Token string `json:"token,omitempty"`

    // Role granted to users who join via this link
    // Always "subuser" - admins must be invited directly
    Role string `json:"role,omitempty"`

    // Maximum number of times this link can be used
    // 0 = unlimited uses
    MaxUses int `json:"max_uses,omitempty"`

    // How many times the link has been used
    CurrentUses int `json:"current_uses,omitempty"`

    // When the link expires (nil = never expires)
    ExpiresAt *time.Time `json:"expires_at,omitempty"`

    // Whether the link is currently active (can be revoked by admin)
    IsActive bool `json:"is_active,omitempty"`

    // Metadata
    CreatedAt time.Time `json:"created_at,omitempty"`
    CreatedBy string    `json:"created_by,omitempty"`
}
```

Validation logic when using a share link:

```
func ValidateShareLink(link *ShareLink) error {
    if !link.IsActive {
        return errors.New("link has been revoked")
    }
    if link.ExpiresAt != nil && time.Now().After(*link.ExpiresAt) {
        return errors.New("link has expired")
    }
    if link.MaxUses > 0 && link.CurrentUses >= link.MaxUses {
        return errors.New("link usage limit reached")
    }
    return nil
}
```

WorkspaceMember

A helper struct for representing a member with their role (used in API responses).

```
type WorkspaceMember struct {
    // The user node
    User *Node `json:"user,omitempty"`

    // Their role in this workspace: "admin" or "subuser"
    Role string `json:"role,omitempty"`

    // When they joined the workspace
    JoinedAt time.Time `json:"joined_at,omitempty"`

    // Who invited them (if known)
    InvitedBy string `json:"invited_by,omitempty"`
}
```

API Reference

Invitation Endpoints

POST /api/workspaces/{id}/invite

Description: Invite an existing user to join the workspace.

Authentication: Required (JWT)

Authorization: Admin only

Path Parameters:

Parameter	Type	Description
id	string	Workspace namespace (e.g., `group_<uuid>`)

Request Body:

```
{
  "username": "alice",
  "role": "subuser"
}
```

Response (201 Created):

```
{
  "invitation_id": "0x1234",
  "status": "pending",
  "message": "Invitation sent to alice"
}
```

Error Responses:

Status	Reason
400	Invalid request body or role
401	Not authenticated
403	Not an admin of this workspace
404	Workspace or user not found
409	User is already a member

GET /api/invitations

Description: List all pending invitations for the current user.

Authentication: Required

Response (200 OK):

```
{
  "invitations": [
    {
      "invitation_id": "0x1234",
      "workspace_id": "group_abc123",
      "workspace_name": "Engineering Team",
      "role": "subuser",
      "invited_by": "bob",
      "created_at": "2024-12-16T10:00:00Z"
    }
  ]
}
```

POST /api/invitations/{id}/accept

Description: Accept a pending invitation.

Authentication: Required

Authorization: Must be the invitee

Response (200 OK):

```
{
  "status": "accepted",
  "workspace_id": "group_abc123",
  "role": "subuser"
}
```

Side Effects:

- User is added to workspace with the specified role
- Invitation status changes to “accepted”

POST /api/invitations/{id}/decline

Description: Decline a pending invitation.

Authentication: Required

Authorization: Must be the invitee

Response (200 OK):

```
{
  "status": "declined"
}
```

Share Link Endpoints

POST /api/workspaces/{id}/share-link

Description: Generate a shareable link for the workspace.

Authentication: Required

Authorization: Admin only

Request Body:

```
{
  "max_uses": 10,
  "expires_in_hours": 72
}
```

Field	Type	Required	Description
max_uses	int	No	Maximum uses (0 or omit for unlimited)
expires_in_hours	int	No	Hours until expiry (0 or omit for never)

Response (201 Created):

```
{
  "token": "a1b2c3d4e5f6g7h8i9j0...",
  "url": "https://your-app.com/join/a1b2c3d4e5f6g7h8i9j0...",
  "max_uses": 10,
  "expires_at": "2024-12-19T10:00:00Z"
}
```

POST /api/join/{token}

Description: Join a workspace using a share link.

Authentication: Required (user must be logged in)

Path Parameters:

Parameter	Type	Description
token	string	The share link token

Response (200 OK):

```
{
  "status": "joined",
  "workspace_id": "group_abc123",
  "workspace_name": "Engineering Team",
  "role": "subuser"
}
```

Error Responses:

Status	Reason
400	Link expired, revoked, or usage limit reached
401	Not authenticated
404	Invalid token
409	Already a member

DELETE /api/workspaces/{id}/share-link/{token}

Description: Revoke a share link.

Authentication: Required

Authorization: Admin only

Response (200 OK):

```
{
  "status": "revoked"
}
```

Member Management Endpoints

GET /api/workspaces/{id}/members

Description: List all members of a workspace.

Authentication: Required

Authorization: Must be a member

Response (200 OK):

```
{
  "members": [
    {
      "username": "bob",
      "role": "admin",
      "joined_at": "2024-12-01T00:00:00Z"
    },
    {
      "username": "alice",
      "role": "subuser",
      "joined_at": "2024-12-15T00:00:00Z",
      "invited_by": "bob"
    }
  ]
}
```

DELETE /api/workspaces/{id}/members/{username}

Description: Remove a member from the workspace.

Authentication: Required

Authorization: Admin only

Response (200 OK):

```
{
  "status": "removed",
  "username": "alice"
}
```

DGraph Schema

New Type Definitions

```
# Workspace invitations for username-based invites
type WorkspaceInvitation {
    workspace_id      # Group namespace
    invitee_user_id   # Username of invitee
    role              # "admin" or "subuser"
    status            # "pending", "accepted", "declined"
    created_at
    created_by
}

# Shareable links for quick access
type ShareLink {
    workspace_id      # Group namespace
    token             # Cryptographic token
    role              # Always "subuser"
    max_uses          # Usage limit
    current_uses      # Current count
    expires_at        # Expiry timestamp
    is_active         # Can be revoked
    created_at
    created_by
}
```

New Predicates

```
# Collaboration predicates (all indexed for fast queries)
workspace_id: string @index(exact) .     # For finding invites/links by workspace
invitee_user_id: string @index(exact) .  # For finding invites for a user
token: string @index(exact) .            # For looking up share links
max_uses: int .                          # Usage limit
current_uses: int .                      # Current usage count
is_active: bool @index(bool) .           # Active/revoked status
role: string @index(exact) .             # For filtering by role
```

Existing Predicates (Reused)

```
# These already exist in the schema
status: string @index(exact) .
created_at: datetime @index(hour) .
created_by: string @index(exact) .
expires_at: datetime .
```

User Flows

Flow 1: Inviting a Team Member

Scenario: Bob (admin) wants to invite Alice (existing user) to the Engineering workspace.

Flow 2: Using a Share Link

Scenario: Bob (admin) wants to quickly onboard new team members using a link.

```
┌─────────────────────────────────────────────────────────────┐
│ Step 1: Bob creates a share link                            │
│ POST /api/workspaces/group_abc123/share-link                │
│ Body: { "max_uses": 5, "expires_in_hours": 48 }             │
│                                                             │
│ Response:                                                   │
│ { "token": "xYz123...",                                     │
│   "url": "https://app.com/join/xYz123...",                  │
│   "expires_at": "2024-12-18T10:00:00Z" }                    │
└─────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────┐
│ Step 2: Bob shares the URL via Slack/Email                  │
│ "Hey team, join our workspace: https://app.com/join/xYz123..."  │
└─────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────┐
│ Step 3: Charlie clicks the link                             │
│                                                             │
│ If not logged in:                                           │
│   → Redirected to login page                                │
│   → After login, continues to join                          │
│                                                             │
│ If logged in:                                               │
│   → Proceeds directly to join                               │
└─────────────────────────────────────────────────────────────┘
                            │
                            ▼
┌─────────────────────────────────────────────────────────────┐
│ Step 4: System processes join request                       │
│ POST /api/join/xYz123...                                    │
│                                                             │
│ Validation:                                                 │
│ 1. Token exists ✓                                           │
│ 2. Link is active (not revoked) ✓                           │
│ 3. Link not expired ✓                                       │
│ 4. Usage limit not reached (2/5 uses) ✓                     │
│ 5. User not already a member ✓                              │
│                                                             │
│ Actions:                                                    │
│ 1. Increment current_uses (2 → 3)                           │
│ 2. Add Charlie as subuser to workspace                      │
│ 3. Return success                                           │
└─────────────────────────────────────────────────────────────┘
```

Flow 3: Admin Spawning Subusers

Scenario: Bob (admin) wants to create a new user account and add them to the workspace immediately.

```
┌─────────────────────────────────────────────────────────────┐
│ POST /api/groups/group_abc123/subusers                      │
│ Body: { "username": "david", "password": "temp123" }        │
│                                                             │
│ System:                                                     │
│ 1. Creates new user "david" in Redis (auth store)           │
│ 2. Creates User node in DGraph                              │
│ 3. Adds david as subuser to workspace                       │
│ 4. Returns credentials                                      │
│                                                             │
│ Response:                                                   │
│ { "status": "subuser_created", "username": "david" }        │
└─────────────────────────────────────────────────────────────┘
```

Security Model

Authentication Requirements

All collaboration endpoints require JWT authentication:

```
// Middleware validates JWT on every request
func (m *JWTMiddleware) Middleware(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        authHeader := r.Header.Get("Authorization")
        if authHeader == "" || !strings.HasPrefix(authHeader, "Bearer ") {
            http.Error(w, "Unauthorized", http.StatusUnauthorized)
            return
        }
        // ... validate token ...
    })
}
```

Share Link Token Security

Tokens are generated using cryptographically secure random bytes:

```
import "crypto/rand"
import "encoding/base64"

func GenerateToken() string {
    bytes := make([]byte, 32) // 256 bits
    rand.Read(bytes)
    return base64.URLEncoding.EncodeToString(bytes)
}
```

Properties:

- 32 bytes = 256 bits of entropy
- Base64URL encoded (URL-safe)
- Practically impossible to guess
- Example: `a1B2c3D4e5F6g7H8i9J0k1L2m3N4o5P6q7R8s9T0u1V2w3X4`

Namespace Isolation

All memory queries enforce namespace filtering:

```
# DGraph query with namespace isolation
query GetFacts($namespace: string) {
    facts(func: type(Fact)) @filter(eq(namespace, $namespace)) {
        name
        description
        # ...
    }
}
```

This ensures:

- User A's workspace cannot see memories from User B's workspace
- Even if the same entities exist, they are isolated by namespace

Audit Trail

All invitations and share links track:

- **created_by**: Who created the invitation/link
- **created_at**: When it was created
- **status**: Current state (for invitations)
- **current_uses**: How many times used (for share links)

Implementation Details

Files Modified

File	Changes
`internal/graph/schema.go`	Added `WorkspaceInvitation`, `ShareLink`, `WorkspaceMember` types
`internal/graph/client.go`	Added DGraph schema types and predicates
`internal/agent/server.go`	New API route handlers (pending implementation)

New Graph Client Functions

```
// Invitation management
InviteToWorkspace(ctx, workspaceNS, inviterID, inviteeUsername, role string) (*WorkspaceInvitation, error)
AcceptInvitation(ctx, invitationID, userID string) error
DeclineInvitation(ctx, invitationID, userID string) error
GetPendingInvitations(ctx, userID string) ([]WorkspaceInvitation, error)

// Share link management
CreateShareLink(ctx, workspaceNS, creatorID string, maxUses int, expiresAt *time.Time) (*ShareLink, error)
JoinViaShareLink(ctx, token, userID string) error
RevokeShareLink(ctx, token, userID string) error

// Member management
GetWorkspaceMembers(ctx, workspaceNS string) ([]WorkspaceMember, error)
```

Testing Guide

Prerequisites

1. Monolith service running (`./monolith.exe` or `docker-compose up`)
2. DGraph accessible at port 8180
3. Python with `requests` library

Test Script

Create `test_workspace_collaboration.py`:

```
import requests
import uuid

BASE_URL = "http://localhost:3000/api"

def test_invite_flow():
    """Test complete invitation flow"""
    # 1. Register admin and invitee
    admin = f"admin_{uuid.uuid4().hex[:6]}"
    invitee = f"invitee_{uuid.uuid4().hex[:6]}"

    admin_token = register(admin, "pass123")
    invitee_token = register(invitee, "pass123")

    # 2. Admin creates workspace
    workspace = create_group(admin_token, "Test Workspace")

    # 3. Admin invites invitee
    invite_resp = invite_user(admin_token, workspace["namespace"], invitee)
    assert invite_resp.status_code == 201

    # 4. Invitee accepts
    invitations = get_invitations(invitee_token)
    assert len(invitations) == 1

    accept_resp = accept_invitation(invitee_token, invitations[0]["id"])
    assert accept_resp.status_code == 200

    # 5. Verify invitee can access workspace
    members = get_members(admin_token, workspace["namespace"])
    assert invitee in [m["username"] for m in members]

    print("✅ Invite flow test passed!")

def test_share_link_flow():
    """Test share link join flow"""
    # ... similar structure ...
    print("✅ Share link flow test passed!")

if __name__ == "__main__":
    test_invite_flow()
    test_share_link_flow()
```

Running Tests

```
cd c:\Users\Akash Kesav\Documents\Whitepaper
python test_workspace_collaboration.py
```

Expected Output

```
✅ Invite flow test passed!
✅ Share link flow test passed!
```

Future Enhancements

Enhancement	Description	Priority
Email invitations	Send invites to emails, create pending entries	Medium
Role granularity	Add “viewer” (read-only) and “commenter” roles	Low
Invitation expiry	Auto-expire unclaimed invitations after N days	Low
Transfer ownership	Allow admin to transfer ownership to another admin	Medium
Activity log	Track all collaboration events for audit	Medium

Last Updated: December 2024