Integrating Shared Memory with ChatGPT

Overview

ChatGPT can access your shared memory system through **Custom GPT Actions**. This requires exposing your backend API and creating an OpenAPI schema.

Step 1: Expose Your Backend API

Your FastAPI backend needs to be accessible from the internet. You have 3 options:

Option A: Use ngrok (Easiest for Testing)

```bash
# Install ngrok
brew install ngrok  # Mac
# or download from https://ngrok.com

# Start your backend
python backend.py

# In another terminal, expose it
ngrok http 8000
```

You'll get a URL like: `https://abc123.ngrok-free.app`

Option B: Deploy to Cloud (Production)

Deploy to Railway, Render, or Heroku:

```bash
# Example with Railway
railway init
railway up
```

Option C: Use Cloudflare Tunnel

```bash
cloudflare tunnel --url localhost:8000
```

Step 2: Create OpenAPI Schema

Save this as `openapi.yaml`:

```
yaml
```

```yaml
openapi: 3.1.0
info:
  title: Shared Memory API
  description: Store and retrieve memories across AI assistants
  version: 1.0.0
servers:
  - url: https://your-ngrok-url.ngrok-free.app
    description: Production server

paths:
  /memory/add:
    post:
      operationId: addMemory
      summary: Add a new memory
      description: Store a new memory for a specific project
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required:
                - project
                - content
              properties:
                project:
                  type: string
                  description: Project identifier (e.g., 'chatgpt', 'claude', 'cursor')
                content:
                  type: string
                  description: The memory content to store
                tags:
                  type: array
                  items:
                    type: string
                  description: Optional tags for categorization
      responses:
        '200':
          description: Memory added successfully
          content:
            application/json:
              schema:
                type: object
              properties:
                success:
                  type: boolean
                memory:
                  type: object

/memory/search:
  get:
    operationId: searchMemory
    summary: Search memories
    description: Search for memories by query string
    parameters:
      - name: query
        in: query
        required: true
        schema:
          type: string
        description: Search query
      - name: limit
        in: query
        schema:
          type: integer
          default: 10
        description: Maximum number of results
    responses:
      '200':
        description: Search results
        content:
          application/json:
            schema:
              type: object
              properties:
                success:
                  type: boolean
                count:
                  type: integer
                memories:
                  type: array
                  items:
                    type: object

/memory/list:
  get:
    operationId: listMemories
```

```yaml
      summary: List memories by project
      description: List all memories, optionally filtered by project
      parameters:
        - name: project
          in: query
          schema:
            type: string
          description: Filter by project name
        - name: limit
          in: query
          schema:
            type: integer
            default: 50
          description: Maximum number of results
      responses:
        '200':
          description: List of memories
          content:
            application/json:
              schema:
                type: object
                properties:
                  success:
                    type: boolean
                  count:
                    type: integer
                  memories:
                    type: array

  /memory/update/{memory_id}:
    put:
      operationId: updateMemory
      summary: Update a memory
      description: Update an existing memory by ID
      parameters:
        - name: memory_id
          in: path
          required: true
          schema:
            type: string
          description: Memory ID to update
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                content:
                  type: string
                  description: New content
                tags:
                  type: array
                  items:
                    type: string
                  description: New tags
      responses:
        '200':
          description: Memory updated
        '404':
          description: Memory not found

  /memory/delete/{memory_id}:
    delete:
      operationId: deleteMemory
      summary: Delete a memory
      description: Delete an existing memory by ID
      parameters:
        - name: memory_id
          in: path
          required: true
          schema:
            type: string
          description: Memory ID to delete
      responses:
        '200':
          description: Memory deleted
        '404':
          description: Memory not found
```

Step 3: Create Custom GPT

1. **Go to ChatGPT** → https://chat.openai.com/gpts/editor
2. **Click "Create a GPT"**
3. **Configure Basic Info:**

- **Name**: "Shared Memory Assistant"
- **Description**: "Access and store memories across AI assistants"
- **Instructions**:

```
  You are a memory management assistant that can store and retrieve information across multiple AI platforms. When
users ask you to remember something, use the addMemory action. When they want to recall information, use
searchMemory or listMemories. Always tag memories with relevant keywords for easy retrieval.

  When storing memories:
  - Always use "chatgpt" as the project identifier
  - Extract key information and store it concisely
  - Add relevant tags for categorization

  When retrieving memories:
  - Search broadly first, then narrow down
  - Present results in a clear, organized format
```

4. **Add Actions:**
   - Click "Create new action"
   - Paste your OpenAPI schema
   - Authentication: **None** (for local testing) or **API Key** (for production)
5. **Configure Privacy:**
   - Add privacy policy URL (required for publishing)
   - Choose: "Only me", "Anyone with a link", or "Public"
6. **Test It:**

```
  "Remember that I prefer TypeScript for my chatgpt projects"
  "What do I prefer for my projects?"
```

Step 4: Add Authentication (Optional but Recommended)

For production, add API key authentication:

In your FastAPI backend:

```
python
```

```
from fastapi import Header, HTTPException

API_KEY = "your-secret-key-here"

async def verify_api_key(x_api_key: str = Header()):
    if x_api_key != API_KEY:
        raise HTTPException(status_code=401, detail="Invalid API key")
    return x_api_key

# Add to each endpoint
@app.post("/memory/add", dependencies=[Depends(verify_api_key)])
async def add_memory(memory: Memory):
    # ... existing code
```

In Custom GPT Actions:

- Authentication Type: **API Key**
- Auth Type: **Custom**
- Custom Header Name: `X-API-Key`
- API Key: `your-secret-key-here`

Step 5: Test Integration

Try these prompts in your Custom GPT:

```
"Remember: I'm working on a React project with TypeScript"

"What projects am I working on?"

"Search for anything related to TypeScript"

"Show me all my chatgpt memories"
```

Troubleshooting

Issue: "Failed to fetch"

- Check if your backend is running

- Verify ngrok URL is correct
- Check CORS settings in FastAPI

Issue: "Authentication failed"

- Verify API key is correct
- Check header name matches

Issue: GPT doesn't call actions

- Make sure OpenAPI schema is valid
- Check operation IDs are unique
- Verify authentication is set correctly

Next: Share with Other AIs

Once working, you can:

1. Use the same API from Gemini (see Gemini guide)
2. Create VS Code extension
3. Build Cursor integration
4. Connect to other tools

Your ChatGPT can now share memories with Claude! 🎉