

Table of Contents

- 1. [System Overview](#)
- 2. [Architecture](#)
- 3. [Technology Stack](#)
- 4. [Data Flow](#)
- 5. [Database Design](#)
- 6. [API Reference](#)
- 7. [User Flow](#)
- 8. [Security](#)
- 9. [Deployment](#)
- 10. [Development](#)
- 11. [Troubleshooting](#)

---

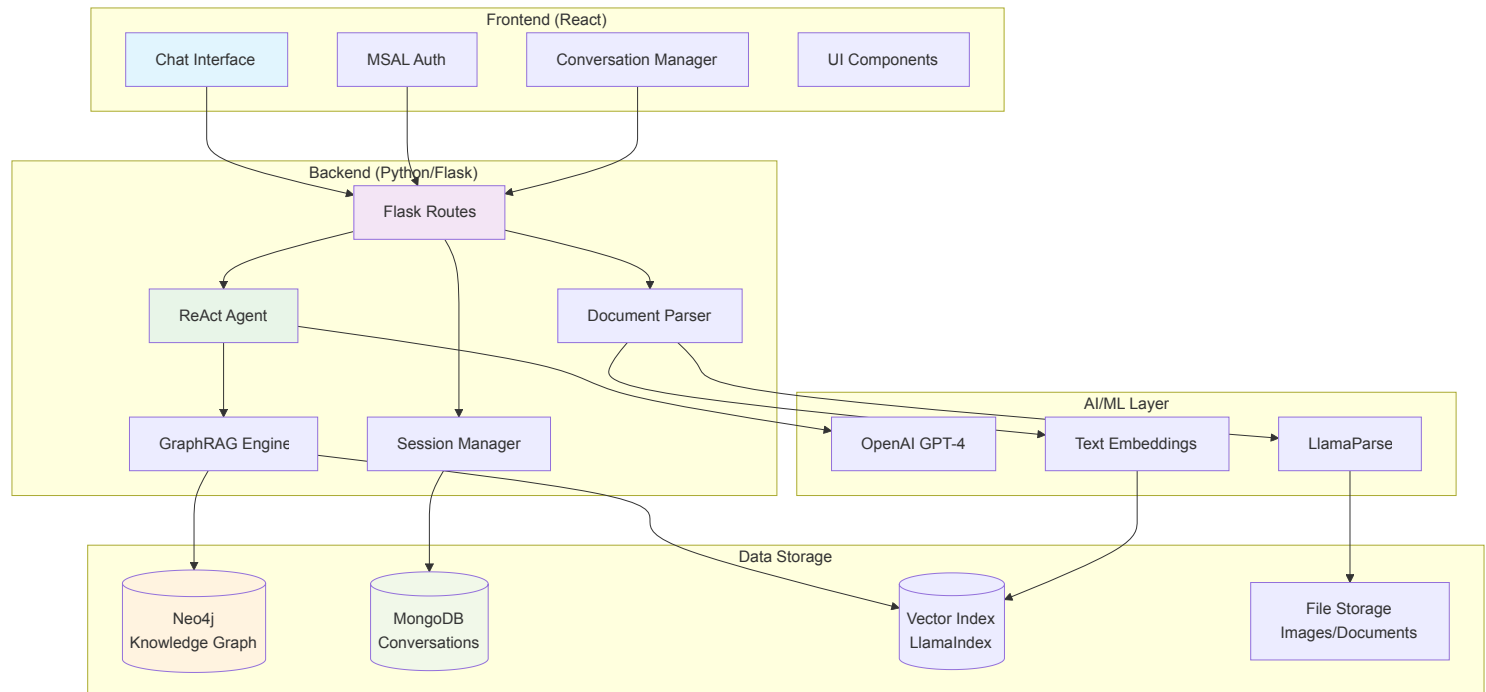
System Overview

The HP GraphRAG Chatbot is a sophisticated conversational AI system that combines vector search with knowledge graph capabilities to answer questions about HP marketing materials and brand guidelines. It processes multimodal documents (text + images) and uses a hybrid AI agent approach for intelligent information retrieval and response generation.

Key Features

- **Multimodal Document Processing:** Extracts text and images from PDFs, PowerPoint, and other marketing documents
  - **GraphRAG Architecture:** Combines vector similarity search with knowledge graph community detection
  - **Custom ReAct Agent:** Implements reasoning and action patterns for intelligent query processing
  - **Session Management:** Maintains conversation context across multiple interactions
  - **Image Display:** Shows relevant document screenshots alongside responses
  - **Authentication:** Microsoft Authentication Library (MSAL) integration
  - **Conversation History:** Persistent storage and retrieval of chat sessions
- 

Architecture



## Component Breakdown

### Frontend (React)

- **Chat Interface:** Main conversational UI with message bubbles, image viewing, and input handling
- **Authentication:** MSAL-based Microsoft authentication
- **Conversation Manager:** Handles multiple conversation sessions and history
- **Theme Toggle:** Dark/light mode support

### Backend (Python/Flask)

- **Flask Routes:** RESTful API endpoints for chat, authentication, file serving
- **ReAct Agent:** Custom implementation with reasoning, action, and observation cycles
- **GraphRAG Engine:** Hybrid retrieval combining vector search with graph-based community detection
- **Session Manager:** Maps frontend sessions to database conversations
- **Document Parser:** LlamaParse integration for multimodal document processing

### Data Layer

- **Neo4j:** Stores knowledge graph with entities, relationships, and communities
- **MongoDB:** Persists user conversations, messages, and session state
- **Vector Index:** LlamaIndex-based semantic search capabilities
- **File Storage:** Local filesystem for processed images and documents

## Technology Stack

### Backend

- **Framework:** Flask + Hypercorn (ASGI)
- **AI/ML:**
  - OpenAI GPT-4 (LLM)
  - text-embedding-3-small (embeddings)
  - LlamaParse (document processing)

- LlamaIndex (vector indexing)
- **Databases:**
  - Neo4j (knowledge graph)
  - MongoDB (conversations)
- **Languages:** Python 3.9+

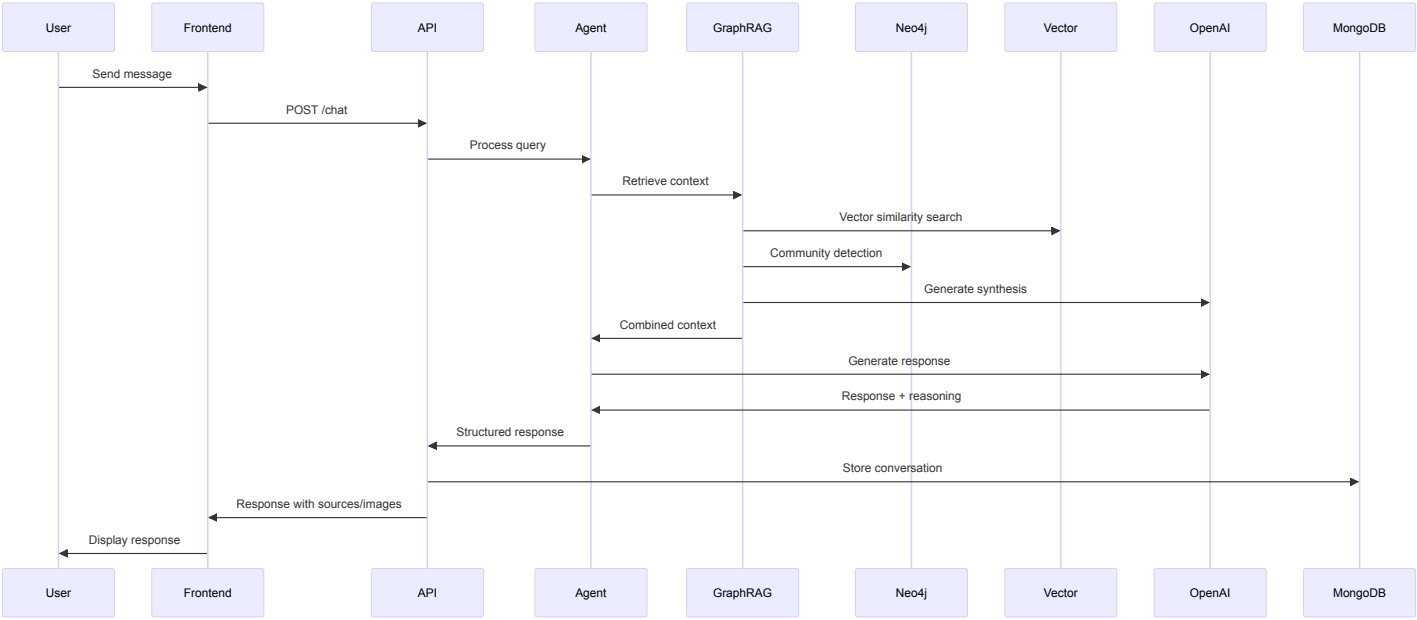
Frontend

- **Framework:** React 18 + Vite
- **Styling:** TailwindCSS + Shadcn/ui
- **Authentication:** Microsoft Authentication Library (MSAL)
- **Languages:** JavaScript/JSX

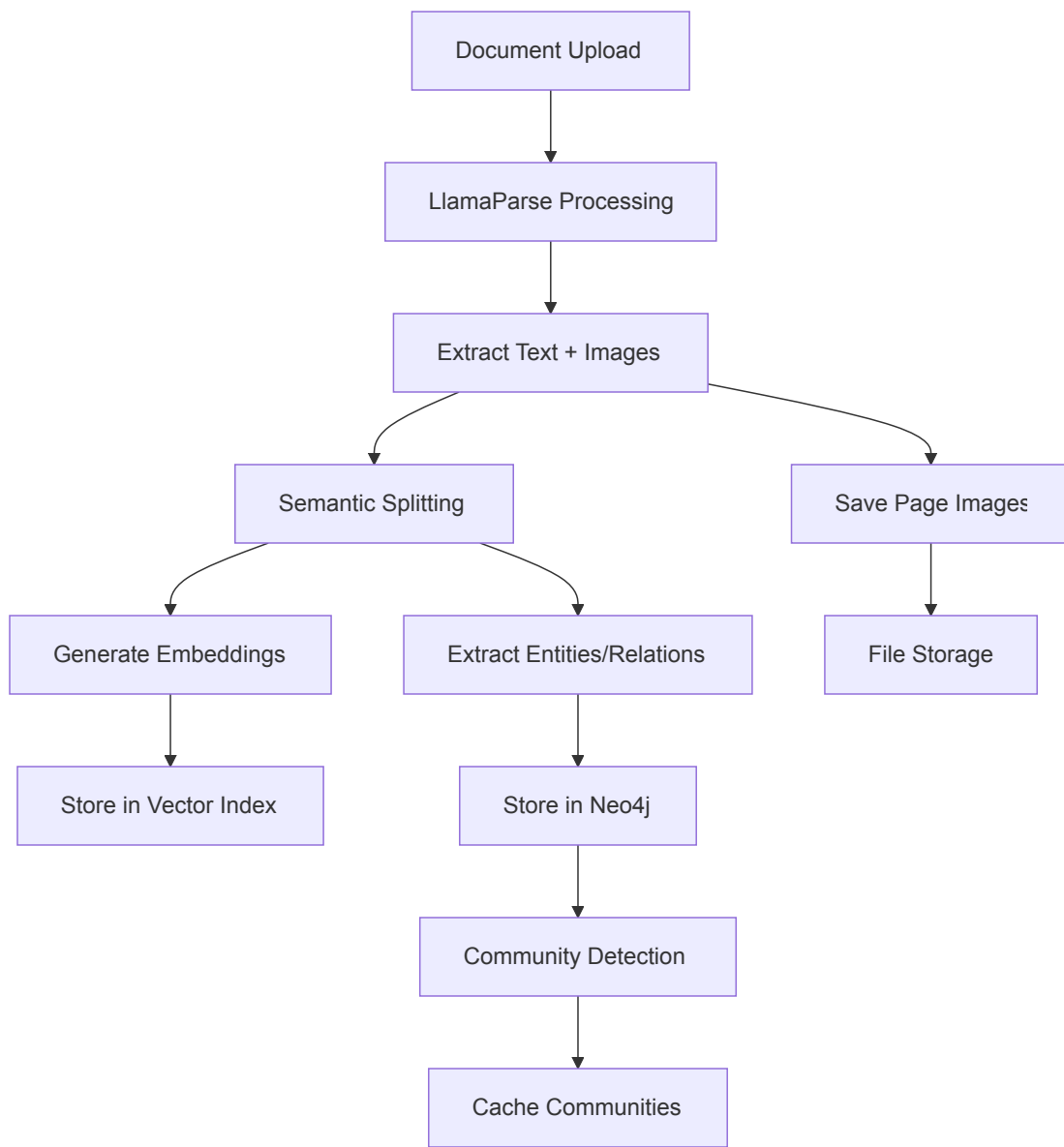
Infrastructure

- **Web Server:** Hypercorn (ASGI server)
- **Containerization:** Docker support
- **Deployment:** Azure/Cloud-based

Data Flow



Document Processing Flow



---

## Database Design

### Neo4j Knowledge Graph Schema

| Community |              |
|-----------|--------------|
| int       | community_id |
| text      | summary      |
| list      | entity_ids   |

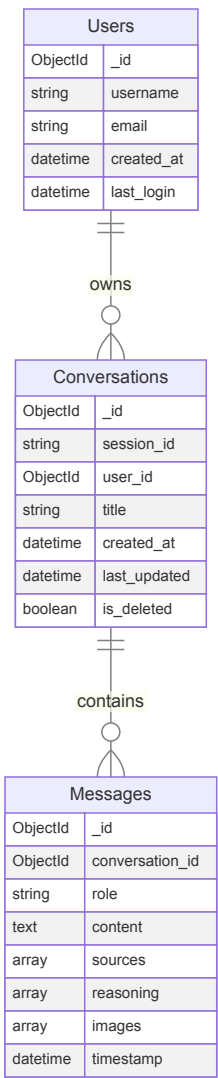
contains

| Entity |             |
|--------|-------------|
| string | name        |
| string | label       |
| string | description |
| dict   | properties  |

participates\_in

| Relation |             |
|----------|-------------|
| string   | label       |
| string   | source_id   |
| string   | target_id   |
| string   | description |
| dict     | properties  |

MongoDB Collections Schema



## API Reference

### Authentication

All API endpoints require authentication via `X-MS-USERNAME` header (except in development mode).

### Core Endpoints

#### POST /chat

Processes chat messages and returns AI responses.

#### Request:

Json ▾

```
1 {
2   "message": "string",
3   "sessionId": "string"
4 }
```

#### Response:

```
1  {
2    "status": "success",
3    "data": {
4      "response": "string",
5      "sources": [
6        {
7          "content": "string",
8          "tool_name": "string",
9          "retrieval_method": "vector_only|graphrag_hybrid"
10       }
11     ],
12     "reasoning": [
13       {
14         "type": "ActionReasoningStep|ObservationReasoningStep",
15         "action": "string",
16         "observation": "string"
17       }
18     ],
19     "images": [
20       {
21         "filename": "string",
22         "document": "string",
23         "page": "number",
24         "url_encoded_filename": "string"
25       }
26     ]
27   }
28 }
```

GET /status

Returns system initialization status.

#### Response:

```
1  {
2    "global_status": "initialized",
3    "initialized": true,
4    "timestamp": "2024-01-01T00:00:00.000Z"
5  }
```

GET /conversations

Retrieves user's conversation history.

#### Response:

```
1  {
2    "status": "success",
3    "conversations": [
4      {
5        "id": "string",
6        "title": "string",
7      }
8    ]
9  }
```

```
7      "created_at": "datetime",
8      "last_updated": "datetime",
9      "session_id": "string"
10    }
11  ]
12 }
```

GET /conversations/{id}/messages

Retrieves messages for a specific conversation.

#### Response:

Json ▾

```
1  {
2    "status": "success",
3    "conversation_title": "string",
4    "messages": [
5      {
6        "id": "string",
7        "role": "user|assistant",
8        "content": "string",
9        "timestamp": "datetime",
10       "sources": [],
11       "reasoning": [],
12       "images": []
13     }
14   ]
15 }
```

GET /images/{filename}

Serves processed document images.

POST /reset

Resets the global agent's conversation memory.

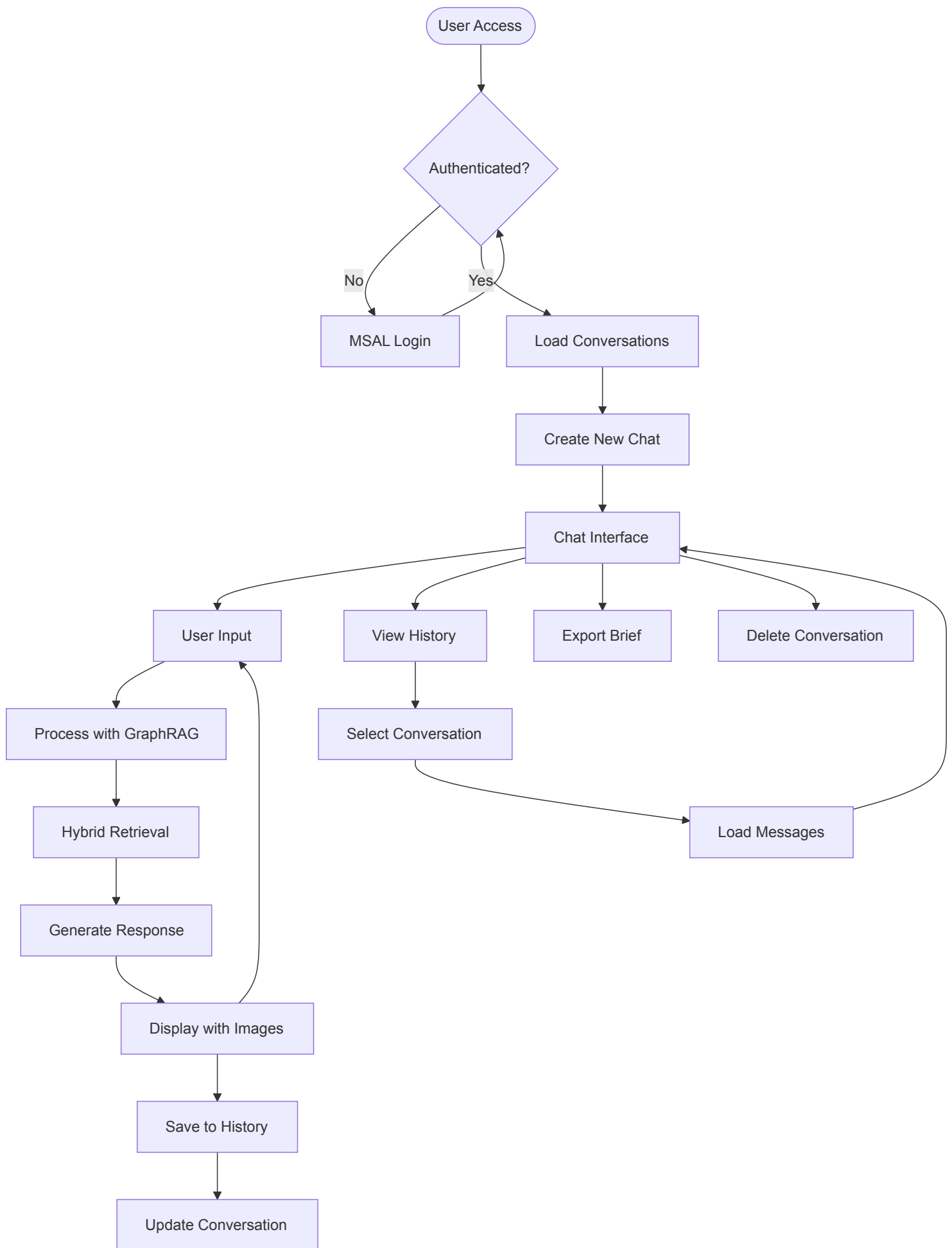
DELETE /conversations/{id}

Deletes a conversation (soft delete by default).

---

## User Flow





1. **Authentication:** User logs in via Microsoft MSAL
  2. **Conversation Creation:** System creates new conversation or loads existing
  3. **Query Processing:**
    - User sends message
    - GraphRAG performs hybrid retrieval
    - Vector similarity search finds relevant chunks
    - Knowledge graph identifies related communities
    - LLM synthesizes response with reasoning
  4. **Response Display:**
    - Text response with markdown support
    - Source attribution tooltips
    - Relevant document images
    - Reasoning chain (if available)
  5. **History Management:** Conversations persisted and retrievable
- 

## Security

### Authentication

- Microsoft Authentication Library (MSAL) integration
- Azure AD tenant-based access control
- Session-based user identification

### Data Protection

- No sensitive data logged in plain text
- Conversation data encrypted at rest (MongoDB)
- API key management via environment variables
- CORS configuration for cross-origin requests

### Access Control

- User-scoped conversation access
  - Session-based authorization
  - Development vs production mode differentiation
- 

## Deployment

### Environment Configuration

#### Backend (.env)

Bash ▾

```
1  # API Keys
2  OPENAI_API_KEY=your_openai_key
3  LLAMA_CLOUD_API_KEY=your_llama_cloud_key
4  ANTHROPIC_API_KEY=your_anthropic_key
5
6  # Database Configuration
7  NE04J_URL=bolt://localhost:7688
8  NE04J_USERNAME=neo4j
9
```

```
10 NE04J_PASSWORD=hp-graphrag-2024
11
12 # Server Configuration
13 PORT=8746
14 PRODUCTION=true
LOG_LEVEL=INFO
```

#### Frontend (.env)

```
1 VITE_BACKEND_URL=https://ai-sandbox.oliver.solutions/hp_chatbot_back
2 VITE_APP_BASE_URL=/hp_chatbot/
```

### Deployment Steps

#### 1. Database Setup:

- Neo4j instance on port 7688
- MongoDB with authentication
- Initialize collections via `init_mongodb.py`

#### 2. Backend Deployment:

```
1 pip install -r requirements.txt
2 python main.py
```

#### 3. Frontend Build:

```
1 cd chat-interface
2 npm install
3 npm run build
# Deploy dist/ contents to /hp_chatbot/ path
```

#### 4. Web Server Configuration:

- Configure reverse proxy (nginx/Apache)
- Set up SSL certificates
- Configure CORS origins

---

## Development

### Backend Development

```
1 # Setup virtual environment
2 python -m venv env
3 source env/bin/activate # or env\Scripts\activate on Windows
4
5 # Install dependencies
6 pip install -r requirements.txt
7
8 # Start development server
9 python main.py
```

### Frontend Development

Bash ▾

```
1 cd chat-interface
2 npm install
3 npm run dev
```

Key Development Commands

| Command        | Purpose                       |
|----------------|-------------------------------|
| python main.py | Start backend server          |
| npm run dev    | Start frontend dev server     |
| npm run build  | Build frontend for production |
| npm run lint   | Lint frontend code            |

Code Structure

▾

```
hp_graphRAG_bot/
├── Backend (Python)
│   ├── main.py           # Application entry point
│   ├── ai_core.py        # Core AI engine & ReAct agent
│   ├── graph_rag_integration.py # GraphRAG system
│   ├── routes.py         # Flask API routes
│   ├── session_manager.py # Session management
│   ├── mongodb_utils.py  # MongoDB operations
│   ├── config.py         # Configuration
│   └── shared_state.py    # Global state management
├── Frontend (React)
│   └── chat-interface/
│       ├── src/
│       │   ├── App.jsx    # Main application component
│       │   ├── components/ # React components
│       │   ├── auth.js    # MSAL authentication
│       │   └── lib/        # Utilities
│       └── dist/          # Production build
└── Data Storage
    ├── uploads/images/    # Processed document images
    ├── index_storage/     # Vector index data
    └── supporting_files/   # Source documents
```

Troubleshooting

Common Issues

Backend Issues

**Problem:** Global workflow agent not initialized  
**Solution:** Check OpenAI API key and Neo4j connectivity

```
# Verify environment variables
echo $OPENAI_API_KEY
```

```
# Check Neo4j connection
curl http://localhost:7474
```

**Problem:** LlamaParse timeout during document processing

**Solution:** Increase timeout settings in config.py

```
LLAMA_PARSE_MAX_TIMEOUT = 7200 # 2 hours
```

**Problem:** MongoDB connection failed

**Solution:** Verify MongoDB service and credentials

```
# Check MongoDB status
brew services list | grep mongodb
# Test connection
mongosh mongodb://hp:hp@localhost:27017/hp_chatbot
```

### Frontend Issues

**Problem:** CORS policy blocking requests

**Solution:** Update CORS\_ALLOWED\_ORIGINS in backend config.py

**Problem:** Authentication failures

**Solution:** Verify MSAL configuration and Azure AD settings

**Problem:** Images not loading

**Solution:** Check image file paths and backend /images/ endpoint

### Debug Endpoints

#### Development Mode Only:

- GET /debug-status - System state inspection
- POST /reinitialize - Force agent reinitialization
- POST /capture-screenshot - Manual image extraction testing

### Logging

All components use structured logging:

```
log_structured('info', 'Event description', {'key': 'value'})
```

Log files locations:

- Backend: app.log
- MongoDB operations: mongodb.log

---

## Performance Considerations

### Scaling

- **Vector Index:** Consider PostgreSQL pgvector for large deployments
- **Neo4j:** Implement read replicas for query scaling

- **MongoDB:** Use connection pooling and sharding
- **Caching:** Redis for session and community caches

## Optimization

- **GraphRAG Communities:** Pre-computed and cached
  - **Image Processing:** Async processing with queue system
  - **Memory Management:** Agent memory reset policies
  - **Response Time:** Parallel vector and graph retrieval
- 

## Future Enhancements

### Planned Features

1. **Multi-tenant Architecture:** Support multiple organizations
2. **Advanced Analytics:** Usage metrics and conversation insights
3. **Enhanced Multimodal:** Video and audio processing
4. **Real-time Collaboration:** Multi-user conversations
5. **API Extensions:** Webhook integrations and external tool calling
6. **Advanced Security:** Role-based access control and audit logging

### Technical Debt

- Implement comprehensive test suite
  - Add API rate limiting
  - Improve error handling consistency
  - Optimize database queries
  - Add health check endpoints
- 

*Documentation Version: 1.0*

*Last Updated: 2025-08-18*

*System Version: HP GraphRAG Chatbot v1.0*