API Contracts Documentation

Multi-Agentic Conversational AI System

Base URL

```
http://localhost:8000
```

Content Type

All API requests and responses use `application/json` unless specified otherwise.

---

1. Chat Endpoint

POST `/chat`

Main conversational endpoint with RAG and CRM integration.

Request Schema

```json
{
  "message": "string (required) - User's message/query",
  "user_id": "string (optional) - Unique user identifier",
  "session_id": "string (optional) - Conversation session identifier"
}
```

Response Schema

```json
{
  "response": "string - AI-generated response",
  "user_id": "string - User identifier (generated if not provided)",
  "session_id": "string - Session identifier (generated if not provided)",
  "timestamp": "datetime - Response timestamp",
  "processing_time": "float - Processing time in seconds",
  "rag_sources": ["string"] - List of document sources used",
  "conversation_category": "string - Categorized conversation type"
}
```

Example Request

```bash
curl -X POST "http://localhost:8000/chat" \
  -H "Content-Type: application/json" \
  -d '{
    "message": "What is machine learning?",
    "user_id": "user-123",
    "session_id": "session-456"
  }'
```

Example Response

```json
{
  "response": "Machine learning is a subset of artificial intelligence that enables computers to learn and make deci
  "user_id": "user-123",
  "session_id": "session-456",
  "timestamp": "2024-01-20T10:30:00.123456Z",
  "processing_time": 1.23,
  "rag_sources": ["ai_fundamentals.pdf", "ml_guide.txt"],
  "conversation_category": "information"
}
```

Error Responses

- `400 Bad Request` - Invalid input format
- `500 Internal Server Error` - Processing error

---

2. Document Upload Endpoint

POST `/upload_docs`

Upload documents to populate the RAG knowledge base.

Request Format

- Content-Type: `multipart/form-data`
- Supported formats: PDF, TXT, CSV, JSON
- Multiple files supported

Request Schema

files: File[] (required) - Array of files to upload

Response Schema

json

```json
{
  "uploaded_files": [
    {
      "filename": "string - Original filename",
      "document_id": "string - Generated document ID",
      "status": "string - processed/error"
    }
  ]
}
```

Example Request

bash

```bash
curl -X POST "http://localhost:8000/upload_docs" \
  -F "files=@document1.pdf" \
  -F "files=@document2.txt" \
  -F "files=@data.csv"
```

Example Response

json

```
{
 "uploaded_files": [
  {
   "filename": "document1.pdf",
   "document_id": "doc-uuid-123",
   "status": "processed"
  },
  {
   "filename": "document2.txt",
   "document_id": "doc-uuid-456",
   "status": "processed"
  },
  {
   "filename": "data.csv",
   "document_id": "doc-uuid-789",
   "status": "processed"
  }
 ]
}
```

Error Responses

- `400 Bad Request` - Unsupported file type
- `413 Request Entity Too Large` - File too large
- `500 Internal Server Error` - Processing error

---

3. CRM User Management

POST `/crm/create_user`

Create a new user profile in the CRM system.

Request Schema

json

```
{
  "name": "string (required) - User's full name",
  "email": "string (required) - User's email address",
  "company": "string (optional) - User's company",
  "preferences": {
    "key": "value - User preferences as key-value pairs"
  }
}
```

Response Schema

json

```json
{
  "user_id": "string - Generated unique user ID",
  "message": "string - Success message"
}
```

Example Request

bash

```bash
curl -X POST "http://localhost:8000/crm/create_user" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Doe",
    "email": "john.doe@example.com",
    "company": "Tech Innovations Inc",
    "preferences": {
      "language": "en",
      "timezone": "UTC",
      "notifications": true
    }
  }'
```

Example Response

json

```
{
  "user_id": "user-uuid-123456",
  "message": "User created successfully"
}
```

PUT `/crm/update_user/{user_id}`

Update existing user information.

URL Parameters

- `user_id` (required) - User's unique identifier

Request Schema

```json
{
  "name": "string (optional) - Updated name",
  "email": "string (optional) - Updated email",
  "company": "string (optional) - Updated company",
  "preferences": {
    "key": "value - Updated preferences"
  }
}
```

Response Schema

```json
{
  "message": "string - Success/error message"
}
```

Example Request

```bash
```

```
curl -X PUT "http://localhost:8000/crm/update_user/user-uuid-123456" \
 -H "Content-Type: application/json" \
 -d '{
  "name": "John Smith",
  "company": "New Tech Corp"
 }'
```

Example Response

```json
{
 "message": "User updated successfully"
}
```

Error Responses

- `404 Not Found` - User not found
- `500 Internal Server Error` - Update failed

---

4. Conversation History

GET `/crm/conversations/{user_id}`

Retrieve conversation history for a specific user.

URL Parameters

- `user_id` (required) - User's unique identifier

Query Parameters

- `limit` (optional, default=20) - Maximum number of conversations to return

Response Schema

json

```json
{
  "conversations": [
    {
      "user_id": "string - User identifier",
      "session_id": "string - Session identifier",
      "user_message": "string - User's message",
      "bot_response": "string - AI response",
      "timestamp": "datetime - Message timestamp",
      "category": "string - Conversation category",
      "status": "string - Conversation status"
    }
  ]
}
```

Example Request

bash

```bash
curl -X GET "http://localhost:8000/crm/conversations/user-uuid-123456?limit=10"
```

Example Response

json

```json
{
  "conversations": [
    {
      "user_id": "user-uuid-123456",
      "session_id": "session-uuid-789",
      "user_message": "What is artificial intelligence?",
      "bot_response": "Artificial intelligence is a branch of computer science...",
      "timestamp": "2024-01-20T10:30:00.123456Z",
      "category": "information",
      "status": "active"
    },
    {
      "user_id": "user-uuid-123456",
      "session_id": "session-uuid-789",
      "user_message": "How does machine learning work?",
      "bot_response": "Machine learning works by training algorithms on data...",
      "timestamp": "2024-01-20T10:32:15.654321Z",
      "category": "information",
      "status": "active"
    }
  ]
}
```

Error Responses

- `404 Not Found` - User not found
- `500 Internal Server Error` - Retrieval failed

---

5. Conversation Reset

POST `/reset`

Reset conversation memory globally or for a specific user.

Request Schema

json

```json
{
  "user_id": "string (optional) - Specific user ID to reset"
}
```

Response Schema

json

```json
{
  "message": "string - Success message"
}
```

Example Request (Reset specific user)

bash

```bash
curl -X POST "http://localhost:8000/reset" \
  -H "Content-Type: application/json" \
  -d '{"user_id": "user-uuid-123456"}'
```

Example Request (Reset all)

bash

```bash
curl -X POST "http://localhost:8000/reset" \
  -H "Content-Type: application/json" \
  -d '{}'
```

Example Response

json

```json
{
  "message": "Conversation reset for user user-uuid-123456"
}
```

---

6. Health Check

GET `/health`

Check system health and status.

Response Schema

json

```
{
  "status": "string - System status",
  "timestamp": "datetime - Current timestamp",
  "version": "string - API version"
}
```

Example Request

bash

```bash
curl -X GET "http://localhost:8000/health"
```

Example Response

json

```json
{
  "status": "healthy",
  "timestamp": "2024-01-20T10:30:00.123456Z",
  "version": "1.0.0"
}
```

Common Response Codes

Code	Description
200	Success
400	Bad Request - Invalid input
404	Not Found - Resource not found
413	Request Entity Too Large
422	Unprocessable Entity - Validation error
500	Internal Server Error

Authentication

Currently, the API does not require authentication. In production, consider implementing:

- API key authentication
- JWT tokens
- OAuth 2.0

Rate Limiting

No rate limiting is currently implemented. For production use, implement:

- Request rate limiting per IP/user
- Concurrent request limits
- Resource usage monitoring

Data Types

Conversation Categories

- `support` - Help, support, problem-related queries
- `sales` - Purchase, pricing, cost-related queries
- `information` - General information requests
- `general` - Other conversations

Conversation Status

- `active` - Currently active conversation
- `resolved` - Resolved conversation
- `archived` - Archived/reset conversation

File Types Supported

- `pdf` - PDF documents
- `txt` - Plain text files
- `csv` - Comma-separated values
- `json` - JSON formatted data

Error Handling

All endpoints return errors in the following format:

```json
{
  "detail": "string - Error description"
}
```

Common Error Scenarios

1. **Invalid JSON Format**

   ```json
   {
     "detail": "Invalid JSON format"
   }
   ```

2. **Missing Required Fields**

   ```json
   {
     "detail": "Field 'message' is required"
   }
   ```

3. **File Processing Error**

   ```json
   {
     "detail": "Unsupported file type: .doc"
   }
   ```

4. **Database Connection Error**

   ```json
   {
     "detail": "Database connection failed"
   }
   ```

Testing Examples

Complete Workflow Test

```bash
```

```
# 1. Create a user
USER_RESPONSE=$(curl -s -X POST "http://localhost:8000/crm/create_user" \
  -H "Content-Type: application/json" \
  -d '{"name": "Test User", "email": "test@example.com"}')

USER_ID=$(echo $USER_RESPONSE | jq -r '.user_id')

# 2. Upload documents
curl -X POST "http://localhost:8000/upload_docs" \
  -F "files=@sample.pdf"

# 3. Start conversation
curl -X POST "http://localhost:8000/chat" \
  -H "Content-Type: application/json" \
  -d "{\"message\": \"Hello, I need help with AI\", \"user_id\": \"$USER_ID\"}"

# 4. Continue conversation
curl -X POST "http://localhost:8000/chat" \
  -H "Content-Type: application/json" \
  -d "{\"message\": \"What is machine learning?\", \"user_id\": \"$USER_ID\"}"

# 5. Get conversation history
curl -X GET "http://localhost:8000/crm/conversations/$USER_ID"

# 6. Reset conversation
curl -X POST "http://localhost:8000/reset" \
  -H "Content-Type: application/json" \
  -d "{\"user_id\": \"$USER_ID\"}"
```

Interactive Testing

Visit `http://localhost:8000/docs` for interactive Swagger UI documentation where you can test all endpoints directly from your browser.

Performance Considerations

Response Times

- Chat endpoint: 1-3 seconds (depends on LLM processing)
- Document upload: 5-30 seconds (depends on file size)
- User operations: < 100ms

- Conversation history: < 500ms

Optimization Tips

1. Use connection pooling for MongoDB
2. Implement caching for frequently accessed data
3. Use async operations for better concurrency
4. Consider using a vector database for large-scale RAG

Monitoring and Logging

The API includes basic processing time tracking. For production, implement:

- Request/response logging
- Error tracking
- Performance monitoring
- Usage analytics

This documentation provides comprehensive information for integrating with the Multi-Agentic Conversational AI System API. For additional support or questions, please refer to the main documentation or contact the development team.