API Endpoints Documentation

This document provides comprehensive documentation for all API endpoints in the AI Research Assistant application. The system uses a 3-service architecture with separated responsibilities.

Architecture Overview

- **Frontend (Next.js)**: Port 3000 - User interface
- **Express DB Server**: Port 3001 - ALL database operations via Supabase RPC
- **FastAPI AI Server**: Port 8000 - AI/ML operations and chat functionality ONLY

Base URLs

- **Development**:
  - Express DB Server: `http://localhost:3001`
  - FastAPI AI Server: `http://localhost:8000`
- **Production**:
  - All routes proxied through nginx at `http://bruhmain.3utilities.com`

Authentication

All protected endpoints require JWT authentication via Supabase Auth. Include the token in the Authorization header:

```
Authorization: Bearer <jwt_token>
```

Rate Limiting

PROF

- **API routes**: 10 requests/second (burst: 20)
- **Auth routes**: 5 requests/second (burst: 10)
- **AI routes**: 10 requests/second (burst: 5)

Express DB Server Endpoints (Port 3001)

All database operations are handled by the Express server using Supabase RPC functions.

Authentication Routes

GET /api/auth/status

Get current authentication status.

Response:

```
{
  "authenticated": true,
  "user": {
    "id": "uuid",
    "email": "user@example.com",
    "user_metadata": {},
    "app_metadata": {}
  }
}
```

GET /api/auth/me

Get current user's profile.

Headers: `Authorization: Bearer <token>`

Response:

```
{
  "user_id": 1,
  "auth_user_id": "uuid",
  "email": "user@example.com",
  "first_name": "John",
  "last_name": "Doe",
  "bio": null,
  "phone_number": null,
  "profile_picture_url": null,
  "availability": "available",
  "created_at": "2024-01-01T00:00:00Z",
  "updated_at": "2024-01-01T00:00:00Z"
}
```

PROF

PUT /api/auth/me

Update current user's profile.

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "first_name": "John",
  "last_name": "Doe",
  "bio": "Research scientist",
  "phone_number": "+1234567890",
  "availability": "busy"
}
```

POST /api/auth/sync-profile

Sync profile with Supabase auth data.

Headers: `Authorization: Bearer <token>`

Response:

```
{
  "message": "Profile synced successfully",
  "profile": { /* user profile object */ }
}
```

User Routes

GET /api/users

Get all users (paginated).

Query Parameters:

- `limit` (optional): Number of users to return (default: 100)
- `offset` (optional): Number of users to skip (default: 0)

GET /api/users/:id

Get specific user by ID.

POST /api/users

Create a new user.

PROF

PUT /api/users/:id

Update user information.

DELETE /api/users/:id

Delete a user.

Group Routes

GET /api/groups

Get all groups.

Response:

```
```

```
[
  {
    "group_id": 1,
    "name": "Research Group A",
    "description": "AI research collaboration",
    "created_by": 1,
    "invite_code": "ABC123",
    "is_public": true,
    "created_at": "2024-01-01T00:00:00Z",
    "updated_at": "2024-01-01T00:00:00Z"
  }
]
```

POST /api/groups

Create a new group.

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "name": "New Research Group",
  "description": "Description of the group",
  "is_public": true
}
```

GET /api/groups/:id

Get specific group by ID.

PROF

PUT /api/groups/:id

Update group information.

DELETE /api/groups/:id

Delete a group.

POST /api/groups/:id/join

Join a group using invite code.

Request:

```
{
  "invite_code": "ABC123"
```

```
}
```

POST /api/groups/:id/leave

Leave a group.

GET /api/groups/:id/members

Get group members.

GET /api/groups/user/:userId

Get groups for a specific user.

Session Routes

GET /api/sessions

Get all sessions for the current user.

POST /api/sessions

Create a new session.

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "title": "Research Discussion",
  "description": "Discussing latest AI papers",
  "group_id": 1,
  "status": "active"
}
```

PROF

GET /api/sessions/:id

Get specific session by ID.

PUT /api/sessions/:id

Update session information.

DELETE /api/sessions/:id

Delete a session.

POST /api/sessions/:id/join

Join a session.

POST /api/sessions/:id/leave

Leave a session.

GET /api/sessions/:id/participants

Get session participants.

Message Routes

GET /api/messages/session/:sessionId

Get messages for a specific session.

Query Parameters:

- `limit` (optional): Number of messages to return
- `offset` (optional): Number of messages to skip

Response:

```
[
  {
    "message_id": 1,
    "session_id": 1,
    "sender_id": 1,
    "content": "Hello everyone!",
    "message_type": "text",
    "reply_to": null,
    "sent_at": "2024-01-01T00:00:00Z",
    "sender": {
      "user_id": 1,
      "first_name": "John",
      "last_name": "Doe"
    }
  }
]
```


POST /api/messages

Send a new message.

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "session_id": 1,
  "content": "Hello everyone!",
  "message_type": "text",
  "reply_to": null
}
```

GET /api/messages/:id

Get specific message by ID.

PUT /api/messages/:id

Update a message.

DELETE /api/messages/:id

Delete a message.

Paper Routes

GET /api/papers

Get all papers with optional filtering.

Query Parameters:

- `limit` (optional): Number of papers to return (default: 100)
- `offset` (optional): Number of papers to skip (default: 0)
- `search` (optional): Search term for papers

Response:

PROF

```
[
  {
    "id": 1,
    "paper_id": 1,
    "title": "Attention Is All You Need",
    "authors": "Vaswani et al.",
    "abstract": "The dominant sequence transduction models...",
    "doi": "10.1000/182",
    "arxiv_id": "1706.03762",
    "publication_date": "2017-06-12",
    "venue": "NIPS 2017",
    "url": "https://arxiv.org/abs/1706.03762",
    "file_path": null,
    "metadata": {},
    "created_at": "2024-01-01T00:00:00Z",
    "updated_at": "2024-01-01T00:00:00Z"
```

```
  }
]
```

POST /api/papers

Create a new paper.

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "title": "Paper Title",
  "authors": "Author Names",
  "abstract": "Paper abstract",
  "doi": "10.1000/182",
  "arxiv_id": "1234.5678",
  "publication_date": "2024-01-01",
  "venue": "Conference Name",
  "url": "https://example.com/paper",
  "metadata": {}
}
```

GET /api/papers/:id

Get specific paper by ID.

PUT /api/papers/:id

Update paper information.

DELETE /api/papers/:id

PROF

Delete a paper.

GET /api/papers/:id/related

Get papers related to a specific paper.

Query Parameters:

- `limit` (optional): Number of related papers to return (default: 10)

POST /api/papers/search

Advanced search for papers.

Request:

```
{
  "name": "transformer",
  "tags": ["nlp", "attention"],
  "limit": 20,
  "offset": 0
}
```

POST /api/papers/search/arxiv

Search arXiv papers using the arXiv API.

Request:

```
{
  "query": "attention mechanism",
  "categories": ["cs.AI", "cs.LG"],
  "limit": 20
}
```

Response:

```
[
  {
    "title": "Paper Title",
    "abstract": "Paper abstract",
    "authors": "Author Names",
    "arxiv_id": "1234.5678",
    "categories": ["cs.AI"],
    "primary_category": "cs.AI",
    "published_at": "2024-01-01T00:00:00Z",
    "source_url": "https://arxiv.org/abs/1234.5678",
    "pdf_url": "https://arxiv.org/pdf/1234.5678.pdf",
    "doi": null,
    "journal_ref": null,
    "comment": null
  }
]
```

PROF

POST /api/papers/arxiv

Create a new arXiv paper entry in the database.

Request:

```
```

```
{
  "title": "Paper Title",
  "abstract": "Paper abstract",
  "authors": "Author Names",
  "arxiv_id": "1234.5678",
  "categories": ["cs.AI"],
  "published_at": "2024-01-01T00:00:00Z",
  "source_url": "https://arxiv.org/abs/1234.5678",
  "pdf_url": "https://arxiv.org/pdf/1234.5678.pdf"
}
```

GET /api/papers/sessions/:sessionId

Get papers linked to a specific session.

POST /api/papers/sessions/:sessionId/:paperId

Link a paper to a session.

DELETE /api/papers/sessions/:sessionId/:paperId

Remove paper from session.

Feedback Routes

GET /api/feedback

Get all feedback entries.

POST /api/feedback

Submit new feedback.

PROF

Headers: `Authorization: Bearer <token>`

Request:

```
{
  "session_id": 1,
  "rating": 5,
  "comment": "Great session!",
  "feedback_type": "session"
}
```

GET /api/feedback/:id

Get specific feedback by ID.

PUT /api/feedback/:id

Update feedback.

DELETE /api/feedback/:id

Delete feedback.

Group Chat Routes

GET /api/group-chat/:groupId/messages

Get messages for a group chat.

POST /api/group-chat/:groupId/messages

Send a message to group chat.

GET /api/group-chat/:groupId/participants

Get participants in a group chat.

AI Metadata Routes

GET /api/ai-metadata

Get AI metadata entries.

POST /api/ai-metadata

Create AI metadata entry.

GET /api/ai-metadata/:id

Get specific AI metadata by ID.

PROF

FastAPI AI Server Endpoints (Port 8000)

FastAPI handles ONLY AI/ML operations and chat functionality. It has NO direct database access.

System Routes

GET /health

System health check.

Response:

```
```

```
{
  "status": "healthy",
  "service": "fastapi-ai-server",
  "timestamp": "2024-01-01T00:00:00Z"
}
```

GET /

API root information.

Response:

```
{
  "message": "AI Research Assistant API",
  "version": "1.0.0",
  "docs_url": "/docs"
}
```

Chat Routes

POST /ai/chat/sessions

Create a new chat session.

Response:

```
{
  "session_id": "uuid-session-id"
}
```

PROF

GET /ai/chat/{session_id}/history

Get chat history for a session.

Response:

```
{
  "messages": [
    {
      "role": "user",
      "content": "Hello",
      "timestamp": "2024-01-01T00:00:00Z"
    },
    {
      "role": "assistant",
```

```
      "content": "Hello! How can I help you?",
      "timestamp": "2024-01-01T00:00:00Z"
    }
  ]
}
```

POST /ai/chat/{session_id}

Send a message and get AI response.

Request:

```
{
  "message": "Explain transformer architecture",
  "context": "Research discussion about NLP models"
}
```

Response:

```
{
  "response": "The transformer architecture is a neural network
model...",
  "session_id": "uuid-session-id",
  "model": "groq",
  "timestamp": "2024-01-01T00:00:00Z"
}
```

DELETE /ai/chat/{session_id}

Delete a chat session.

PROF

Response:

```
{
  "message": "Session deleted successfully"
}
```

POST /ai/chat/group-message

Handle AI invocation from group chat.

Request:

```
```

```
{
  "user_message": "Can you explain this paper?",
  "session_id": "uuid-session-id",
  "group_id": 1,
  "context": "Group discussion context"
}
```

Response:

```
{
  "response": "Based on the paper you're discussing...",
  "session_id": "uuid-session-id",
  "model": "groq"
}
```

POST /ai/chat

Legacy endpoint for backward compatibility.

Request:

```
{
  "prompt": "Explain machine learning"
}
```

Response:

```
{
  "response": "Machine learning is a subset of artificial
intelligence..."
}
```

PROF

Error Responses

All endpoints follow consistent error response format:

Standard Error Codes

- **400 Bad Request**: Invalid request parameters
- **401 Unauthorized**: Authentication required or invalid token
- **403 Forbidden**: Insufficient permissions

- **404 Not Found**: Resource not found
- **429 Too Many Requests**: Rate limit exceeded
- **500 Internal Server Error**: Server error
- **503 Service Unavailable**: External service unavailable

Error Response Format

```
{
  "error": "Error message description",
  "code": 400,
  "details": "Additional error details (optional)"
}
```

Common Error Examples

Authentication Error

```
{
  "error": "Authentication required",
  "code": 401
}
```

Validation Error

```
{
  "error": "title and authors are required",
  "code": 400
}
```

PROF

Rate Limit Error

```
{
  "error": "Rate limit exceeded",
  "code": 429
}
```

Service Unavailable

```
{
  "error": "ArXiv service temporarily unavailable",
```

```
  "code": 503
}
```

Database RPC Functions

The Express server uses Supabase RPC functions for database operations. Key functions include:

User Functions

- `get_all_users(p_limit, p_offset)`
- `get_user_by_id(p_user_id)`
- `create_user(...)`
- `update_user(...)`
- `delete_user(p_user_id)`

Group Functions

- `get_all_groups()`
- `create_group(...)`
- `join_group(p_group_id, p_user_id, p_invite_code)`
- `leave_group(p_group_id, p_user_id)`
- `get_user_groups(p_user_id)`

Session Functions

- `get_user_sessions(p_user_id)`
- `create_session(...)`
- `join_session(p_session_id, p_user_id)`
- `get_session_participants(p_session_id)`

PROF

Paper Functions

- `get_all_papers(p_limit, p_offset)`
- `search_papers(p_search_term, p_limit, p_offset)`
- `create_paper(...)`
- `get_session_papers(p_session_id)`
- `add_paper_to_session(p_session_id, p_paper_id)`

Message Functions

- `get_session_messages(p_session_id, p_limit, p_offset)`
- `create_message(...)`
- `update_message(...)`
- `delete_message(p_message_id)`

Development Notes

1. **Service Separation**: FastAPI has ZERO database interactions. All database operations go through Express server.
2. **Authentication**: Supabase JWT tokens are validated by both services but only Express server queries user data.
3. **Communication**: FastAPI uses HTTP client to communicate with Express server for chat persistence.
4. **Rate Limiting**: Implemented at nginx level with different limits for different route types.
5. **Error Handling**: Both services implement consistent error response formats.
6. **CORS**: Configured appropriately for cross-origin requests in development.

For more information, see the project architecture documentation and individual service README files.