AI Tutoring Platform API Documentation

Overview

The AI Tutoring Platform provides RESTful APIs for managing AI teachers, chat sessions, and knowledge bases. The platform supports personality-driven AI teachers with RAG-enhanced responses.

Authentication

All endpoints require a user ID to be passed in the `X-User-ID` header.

```http
X-User-ID: user-123
```

Chat Endpoints

1. Start Chat Session

Endpoint: `POST /chat/start`

Description: Initialize a new chat session with an AI teacher.

Request

Headers:

```http
Content-Type: application/json
X-User-ID: user-123
```

Payload:

```json
{
  "teacher_id": "teacher-uuid-123",
  "title": "Learning Python Basics"
}
```

Payload Schema:

- `teacher_id` (string, required): UUID of the teacher to chat with

- `title` (string, optional): Custom title for the chat session

Response

Success (200):

```json
{
  "id": "chat-uuid-456",
  "user_id": "user-123",
  "teacher_id": "teacher-uuid-123",
  "title": "Learning Python Basics",
  "created_at": "2024-01-15T10:30:00Z",
  "updated_at": "2024-01-15T10:30:00Z",
  "metadata": {}
}
```

Error (400):

```json
{
  "detail": "Failed to start chat. The teacher may not exist."
}
```

---

2. Send Message

Endpoint: `POST /chat/{chat_id}/send`

Description: Send a message to an AI teacher and receive a response with optional RAG enhancement.

Request

Path Parameters:

- `chat_id` (string): UUID of the chat session

Headers:

```http
Content-Type: application/json
X-User-ID: user-123
```

Payload:

```json
{
  "content": "Can you explain how functions work in Python?",
  "metadata": {
    "message_type": "question",
    "difficulty_preference": "beginner",
    "learning_style": "visual"
  }
}
```

Payload Schema:

- `content` (string, required): The message content
- `metadata` (object, optional): Additional context for the message
  - `message_type` (string): Type of message (question, request, etc.)
  - `difficulty_preference` (string): Preferred difficulty level
  - `learning_style` (string): Student's learning style preference

Response

Success (200):

```json
{
  "message_id": "msg-uuid-789",
  "content": "Great question! Functions in Python are reusable blocks of code that per
  "timestamp": "2024-01-15T10:35:00Z",
  "metadata": {
    "teacher_id": "teacher-uuid-123",
    "teacher_name": "Alex Rivera",
    "domain": "Programming",
    "teaching_style": "practical",
    "rag_enhanced": true,
    "sources_used": [
      {
        "title": "Python Functions Guide",
        "source": "Python Documentation",
        "score": 0.92
      }
    ]
  }
}
```

Error (400):

```json
{
  "detail": "Failed to send message or generate response"
}
```

Error (404):

```json
{
  "detail": "Chat session not found"
}
```

---

3. Get Chat History

Endpoint: `GET /chat/{chat_id}/history`

Description: Retrieve the complete message history for a chat session.

Request

Path Parameters:

- `chat_id` (string): UUID of the chat session

Headers:

```http
X-User-ID: user-123
```

Response

Success (200):

json

```json
[
  {
    "id": "msg-system-001",
    "role": "system",
    "content": "You are Alex Rivera, your coding buddy. You have these personality tra
    "timestamp": "2024-01-15T10:30:00Z",
    "metadata": {}
  },
  {
    "id": "msg-user-001",
    "role": "user",
    "content": "Can you explain how functions work in Python?",
    "timestamp": "2024-01-15T10:32:00Z",
    "metadata": {
      "message_type": "question"
    }
  },
  {
    "id": "msg-assistant-001",
    "role": "assistant",
    "content": "Great question! Functions in Python are reusable blocks...",
    "timestamp": "2024-01-15T10:35:00Z",
    "metadata": {
      "teacher_id": "teacher-uuid-123",
      "rag_enhanced": true
    }
  }
]
```

Message Schema:

- `id` (string): Unique message identifier
- `role` (string): Message role - "system", "user", or "assistant"
- `content` (string): Message content
- `timestamp` (string): ISO 8601 timestamp
- `metadata` (object): Additional message metadata

---

4. Get User Chats

Endpoint: `GET /chat/`

Description: Retrieve all chat sessions for the current user, optionally filtered by teacher.

Request

Query Parameters:

- `teacher_id` (string, optional): Filter chats by specific teacher

Headers:

```http
X-User-ID: user-123
```

Response

Success (200):

```json
[
  {
    "id": "chat-uuid-456",
    "user_id": "user-123",
    "teacher_id": "teacher-uuid-123",
    "title": "Learning Python Basics",
    "created_at": "2024-01-15T10:30:00Z",
    "updated_at": "2024-01-15T10:35:00Z",
    "metadata": {}
  },
  {
    "id": "chat-uuid-789",
    "user_id": "user-123",
    "teacher_id": "teacher-uuid-456",
    "title": "Advanced Calculus",
    "created_at": "2024-01-14T14:20:00Z",
    "updated_at": "2024-01-14T15:45:00Z",
    "metadata": {}
  }
]
```

---

5. Rate Message

Endpoint: `POST /chat/{chat_id}/message/{message_id}/rate`

Description: Rate a teacher's response and provide feedback for improvement.

Request

Path Parameters:

- `chat_id` (string): UUID of the chat session
- `message_id` (string): UUID of the message to rate

Headers:

```http
Content-Type: application/json
X-User-ID: user-123
```

Payload:

```json
{
  "rating": 4.5
}
```

Payload Schema:

- `rating` (number, required): Rating from 1-5 (supports decimals)

Response

Success (200):

```json
{
  "message": "Rating submitted successfully",
  "rating": 4.5
}
```

Error (400):

```json
{
  "detail": "Failed to rate message"
}
```

---

6. End Chat Session

Endpoint: `POST /chat/{chat_id}/end`

Description: Mark a chat session as ended.

Request

Path Parameters:

- `chat_id` (string): UUID of the chat session

Headers:

```http
X-User-ID: user-123
```

Response

Success (200):

```json
{
  "message": "Chat session ended successfully"
}
```

Error (400):

```json
{
  "detail": "Failed to end chat session"
}
```

---

7. Get Message Sources

Endpoint: `GET /chat/{chat_id}/message/{message_id}/sources`

Description: Retrieve the sources used for a RAG-enhanced message response.

Request

Path Parameters:

- `chat_id` (string): UUID of the chat session
- `message_id` (string): UUID of the message

Headers:

```http
X-User-ID: user-123
```

Response

Success (200):

```json
{
  "message_id": "msg-uuid-789",
  "rag_enhanced": true,
  "sources": [
    {
      "title": "Python Functions Guide",
      "source": "Python Documentation",
      "score": 0.92,
      "domain": "Programming",
      "difficulty_level": "beginner"
    },
    {
      "title": "Advanced Function Concepts",
      "source": "Programming Textbook",
      "score": 0.87,
      "domain": "Programming",
      "difficulty_level": "intermediate"
    }
  ]
}
```

Error (404):

```json
{
  "detail": "Message not found or not RAG-enhanced"
}
```

Knowledge Base Endpoints

1. Add Single Document

Endpoint: `POST /knowledge-base/document/{teacher_id}`

Description: Add a single document to a teacher's knowledge base.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher

Headers:

```http
Content-Type: application/json
```

Payload:

```json
{
  "title": "Introduction to Python Functions",
  "text": "Functions in Python are defined using the 'def' keyword. A function is a bl
  "source": "Python Tutorial",
  "domain": "Programming",
  "sub_domains": ["Python", "Functions", "Syntax"],
  "difficulty_level": "beginner",
  "tags": ["functions", "python", "basics", "syntax", "tutorial"]
}
```

Payload Schema:

- `title` (string, required): Document title
- `text` (string, required): Document content/text
- `source` (string, optional): Source of the document
- `domain` (string, optional): Primary domain/subject area
- `sub_domains` (array of strings, optional): List of sub-domains
- `difficulty_level` (string, optional): Difficulty level (beginner/intermediate/advanced/expert)
- `tags` (array of strings, optional): List of tags for categorization

Response

Success (200):

```json
{
  "success": true,
  "count": 3,
  "message": "Successfully added 3 document chunks to the knowledge base"
}
```

Error (500):

```json
{
  "detail": "Failed to add document to knowledge base"
}
```

---

2. Add Multiple Documents

Endpoint: `POST /knowledge-base/documents/{teacher_id}`

Description: Add multiple documents to a teacher's knowledge base in batch.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher

Headers:

```http
Content-Type: application/json
```

Payload:

```json
[
  {
    "title": "Python Variables",
    "text": "Variables in Python are created when you assign a value to them. Python h
    "source": "Python Basics Guide",
    "domain": "Programming",
    "sub_domains": ["Python", "Variables"],
    "difficulty_level": "beginner",
    "tags": ["variables", "python", "basics"]
  },
  {
    "title": "Python Data Types",
    "text": "Python has various built-in data types:\n\n1. Text Type: str\n2. Numeric 
    "source": "Python Data Types Reference",
    "domain": "Programming",
    "sub_domains": ["Python", "Data Types"],
    "difficulty_level": "beginner",
    "tags": ["data-types", "python", "basics", "types"]
  }
]
```

Response

Success (200):

```json
{
  "success": true,
  "count": 8,
  "message": "Successfully added 8 document chunks to the knowledge base"
}
```

Error (500):

```json
{
  "detail": "Failed to add documents to knowledge base"
}
```

---

3. Delete Document

Endpoint: `DELETE /knowledge-base/document/{teacher_id}/{document_id}`

Description: Remove a specific document from a teacher's knowledge base.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher
- `document_id` (string): ID of the document to delete

Response

Success (200):

```json
{
  "success": true,
  "message": "Document doc_1642234567_0 deleted successfully"
}
```

Error (404):

```json
{
  "detail": "Document doc_1642234567_0 not found or could not be deleted"
}
```

---

4. Get Collection Information

Endpoint: `GET /knowledge-base/collection/{teacher_id}`

Description: Get information about a teacher's knowledge base collection.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher

Response

Success (200):

```json
{
  "teacher_id": "teacher-uuid-123",
  "document_count": 156,
  "exists": true
}
```

Response Schema:

- `teacher_id` (string): The teacher's UUID
- `document_count` (integer): Number of document chunks in the knowledge base
- `exists` (boolean): Whether the collection exists

---

5. Create Collection

Endpoint: `POST /knowledge-base/collection/{teacher_id}`

Description: Create a new knowledge base collection for a teacher.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher

Response

Success (200):

```json
{
  "success": true,
  "message": "Collection for teacher teacher-uuid-123 created successfully"
}
```

If Already Exists (200):

```json
{
  "success": true,
  "message": "Collection for teacher teacher-uuid-123 already exists"
}
```

Error (500):

```json
{
  "detail": "Failed to create collection for teacher teacher-uuid-123"
}
```

---

6. Delete Collection

Endpoint: `DELETE /knowledge-base/collection/{teacher_id}`

Description: Delete a teacher's entire knowledge base collection.

Request

Path Parameters:

- `teacher_id` (string): UUID of the teacher

Response

Success (200):

```json
{
  "success": true,
  "message": "Collection for teacher teacher-uuid-123 deleted successfully"
}
```

Error (404):

```json
{
  "detail": "Collection for teacher teacher-uuid-123 not found or could not be deleted
}
```

Error Handling

Common HTTP Status Codes

- **200**: Success
- **400**: Bad Request - Invalid input parameters
- **401**: Unauthorized - Missing or invalid authentication
- **404**: Not Found - Resource doesn't exist

- **500**: Internal Server Error - Server-side error

Error Response Format

All error responses follow this format:

```json
{
  "detail": "Descriptive error message"
}
```

Rate Limiting

The API implements rate limiting to prevent abuse:

- **Chat endpoints**: 60 requests per minute per user
- **Knowledge base endpoints**: 30 requests per minute per user
- **Embedding operations**: 5 requests per minute per user (due to external API limits)

When rate limits are exceeded, the API returns:

```json
{
  "detail": "Rate limit exceeded. Please try again later.",
  "retry_after": 60
}
```

Best Practices

1. Document Chunking

- Keep individual documents focused on single topics
- Optimal document length: 500-2000 characters
- Use clear, descriptive titles
- Include relevant metadata and tags

2. Chat Sessions

- End chat sessions when conversations are complete
- Use descriptive titles for better organization
- Rate messages to help improve teacher responses

3. Error Handling

- Always check response status codes
- Implement retry logic for 5xx errors
- Handle rate limiting gracefully

4. Authentication

- Always include the `X-User-ID` header
- Use consistent user identifiers across sessions