1. Core RAG Pipeline

The Core RAG (Retrieval-Augmented Generation) pipeline consists of three essential steps that enable intelligent question-answering using course materials:

1. **Ingestion:** Upload and process documents into searchable vectors
2. **Retrieval:** Search for semantically relevant content chunks
3. **Generation:** Create AI-powered responses using retrieved context

```
User Question → Retrieve Context → Generate Answer ↓ ↓ ↓ (Query) (ChromaDB Search) (GPT-4.1-mini)
```

1.1 Document Ingestion

POST /ingest

Purpose: Upload and process documents (PDFs, URLs, files) for semantic search

Use Cases:

- Professor uploads lecture slides and course materials
- Student adds supplementary resources
- Admin imports institutional content
- Automated content updates via webhooks

Request Format

Option A: File Upload (multipart/form-data)

```
POST /ingest
Content-Type: multipart/form-data

course_name: CS101
file: lecture_01.pdf
readable_filename: Introduction to Programming
```

Option B: URL-based (application/json)

```
POST /ingest
Content-Type: application/json

{
  "course_name": "CS101",
  "url": "https://example.com/lecture.pdf",
  "readable_filename": "Introduction to Programming",
  "base_url": "https://example.com"
}
```

Parameters

Parameter	Type	Required	Description
`course_name`	string	✅ Yes	Unique course identifier (e.g., CS101, BIOL202)
`file`	file	✅*	Document file for file upload method
`url`	string	✅*	Document URL for URL-based ingestion
`readable_filename`	string	❌ No	Display name for document (defaults to filename)
`base_url`	string	❌ No	Base URL for resolving relative links in HTML docs

* Either `file` or `url` must be provided

Response

```
{
  "outcome": "Queued Ingest task",
  "task_id": "abc-123-def-456",
  "success": true
}
```

Processing Flow

> ⏱ **Asynchronous Processing:** The endpoint returns immediately (≈200ms). Background worker processes the document in 30-60 seconds. Use `task_id` for status tracking.

1. **API Layer:** Receives file/URL and validates parameters
2. **Storage:** Saves to temporary location or downloads from URL
3. **Queue:** Creates task in RabbitMQ with metadata
4. **Immediate Response:** Returns task ID to client
5. **Worker Processing (Background):**
    - Downloads file if URL provided
    - Detects file type and selects appropriate parser
    - Extracts text content (with OCR fallback for images)
    - Chunks text: 1000 characters per chunk, 200 character overlap
    - Generates embeddings using Azure OpenAI (text-embedding-3-small)
    - Stores vectors in ChromaDB with metadata
    - Uploads original file to Azure Blob Storage
    - Updates document registry in MS SQL Server
6. **Completion:** Optional email notification sent

Performance Metrics

Stage	Average Time	Notes
API Response	200ms	Immediate return with task ID
File Download (if URL)	3-5s	Depends on file size and network
Text Extraction	2-5s	Varies by file type (PDF slower than TXT)
Text Chunking	1s	Fast for all document sizes
Embedding Generation	10-20s	Depends on chunk count (15-30 chunks typical)
ChromaDB Upload	2-5s	Batch upload of all chunks
Azure Blob Upload	2-4s	Original file backup
Total Processing	**30-60s**	**For typical 10-page PDF**

Supported File Formats

Format Category	Extensions	Processing Method	Special Features
Plain Text	.txt, .md, .py, .js, .java, .cpp	Direct UTF-8 reading	Syntax highlighting preserved in metadata
PDF Documents	.pdf	PyPDF → pdfplumber → OCR fallback	Page numbers extracted, images OCR'd if text layer missing
Office Documents	.docx, .doc	python-docx / textract	Headers, footers, and formatting preserved
Presentations	.pptx, .ppt	python-pptx	Slide numbers and speaker notes included
Spreadsheets	.xlsx, .xls, .csv	pandas / openpyxl	Table structure preserved, formulas evaluated
Images	.png, .jpg, .jpeg, .gif	Tesseract OCR	Handwriting detection, multi-language support
Video	.mp4, .avi, .mov, .mkv	FFmpeg audio extraction → Whisper	Automatic transcription with timestamps
Audio	.mp3, .wav, .m4a, .ogg	Whisper transcription	Speaker diarization if multiple speakers
Web Content	.html, .htm	BeautifulSoup	JavaScript rendered, images extracted

Error Responses

```
// 400 Bad Request - Missing required parameter
{
  "error": "course_name is required",
  "code": "MISSING_PARAMS"
}

// 400 Bad Request - Invalid file
{
  "error": "Empty filename",
  "code": "INVALID_FILE"
}

// 500 Internal Server Error - Processing failure
{
  "error": "Failed to queue ingestion",
  "outcome": "Failed",
  "task_id": "abc-123",
  "details": "RabbitMQ connection timeout"
}
```

Best Practices

✅ DO:

- Use descriptive `readable_filename` values for better search results
- Batch multiple files when possible to reduce API calls
- Verify file size < 100MB before upload (split larger files)
- Use Canvas import for bulk LMS content instead of individual uploads

✕ DON'T:

- Upload duplicate content (wastes embedding credits)
- Use special characters or spaces in course_name
- Expect immediate searchability (wait 30-60s for processing)
- Upload scanned PDFs without OCR preprocessing (degrades quality)

1.2 Context Retrieval

POST /getTopContexts

Purpose: Search for semantically relevant document chunks using vector similarity

Use Cases:

- Find relevant content for student questions before generating AI response
- Research assistant functionality - explore similar topics
- Pre-flight check before chatbot response to ensure relevant context exists
- Manual context verification for quality assurance

Request Format

```
POST /getTopContexts
Content-Type: application/json

{
  "search_query": "What is polymorphism in object-oriented programming?",
  "course_name": "CS101",
  "top_n": 5,
  "doc_groups": ["lectures", "readings"],
  "conversation_id": ""
}
```

Parameters

Parameter	Type	Required	Default	Description
`search_query`	string	✅ Yes	-	Natural language question or search phrase
`course_name`	string	✅ Yes	-	Course identifier to search within
`top_n`	integer	❌ No	5	Number of results to return (1-100)
`doc_groups`	array[string]	❌ No	[]	Filter by document categories/tags
`conversation_id`	string	❌ No	""	Filter to conversation-specific uploaded docs

Response


```
[
  {
    "text": "Polymorphism is one of the four fundamental principles of object-o
    "page_content": "Polymorphism is one of the four fundamental principles..."
    "readable_filename": "Lecture_03_OOP_Concepts.pdf",
    "course_name": "CS101",
    "pagenumber": "12",
    "url": "https://courses.example.edu/cs101/lecture03.pdf",
    "s3_path": "CS101/documents/uuid-abc-123.pdf",
    "score": 0.8947,
    "base_url": "https://courses.example.edu",
    "doc_groups": ["lectures", "week3"]
  },
  {
    "text": "There are two main types of polymorphism in Java: compile-time (me
    "readable_filename": "Java_OOP_Guide.pdf",
    "pagenumber": "45",
    "score": 0.8523,
    "doc_groups": ["textbook"]
  },
  {
    "text": "Example of polymorphism in Python:\n\nclass Animal:\n    def sound
    "readable_filename": "Python_Examples.py",
    "score": 0.8201,
    "doc_groups": ["code_examples"]
  }
]
```

Response Fields

Field	Type	Description
`text`	string	The actual content chunk (typically 800-1200 characters)
`page_content`	string	Duplicate of text field for LangChain compatibility
`readable_filename`	string	Human-friendly document name
`pagenumber`	string	Page number in source document (if applicable)
`score`	float	Cosine similarity score (0-1, higher = more relevant)
`url`	string	Original source URL (if ingested via URL)
`s3_path`	string	Azure Blob Storage path for original file

Processing Flow

1. **Query Reception:** API receives search query and parameters
2. **Query Embedding:**
    - Sends query to Azure OpenAI (text-embedding-3-small)
    - Receives 1536-dimensional vector representation
    - Takes ~1.2 seconds
3. **Vector Search:**
    - Queries ChromaDB with embedding vector
    - Applies filters: course_name, doc_groups, conversation_id
    - Uses cosine similarity for ranking
    - Takes ~0.6 seconds for typical course (500-5000 documents)
4. **Result Processing:**
    - Retrieves top N matching chunks with metadata
    - Formats response with all fields
    - Returns JSON array sorted by relevance score

```
search_query → Azure OpenAI Embedding (1.2s) ↓ 1536-dimensional vector ↓
ChromaDB Similarity Search (0.6s) ↓ Filter by course_name + params ↓
Return top N results ↓ Total time: ~1.9s
```

Performance Metrics

Operation	Average	P50	P95	P99
Query Embedding	1.2s	1.1s	1.8s	2.5s
Vector Search (ChromaDB)	0.6s	0.5s	1.2s	2.0s
Result Processing	0.1s	0.05s	0.2s	0.3s
Total Request Time	**1.9s**	**1.7s**	**3.2s**	**4.8s**

Error Responses

```
// 400 Bad Request - Missing parameters
{
  "error": "Missing required parameters",
  "message": "search_query and course_name must be provided",
  "code": "MISSING_PARAMS"
}

// 404 Not Found - Course doesn't exist
{
  "error": "Course not found",
  "message": "No course with name 'CS999'",
  "code": "COURSE_NOT_FOUND"
}

// 500 Internal Server Error - Embedding failure
{
  "error": "Failed to generate query embedding",
  "message": "Azure OpenAI service unavailable",
  "code": "EMBEDDING_ERROR"
}
```

1.3 AI Response Generation

POST /chat

Purpose: Generate AI-powered answers using retrieved context from course materials

Use Cases:

- Student asks course-related question and gets instant AI tutor response
- Interactive learning with follow-up questions and context awareness
- Teaching assistant automation for common queries
- 24/7 course support with accurate, sourced responses

Key Feature: This endpoint automatically performs retrieval internally. You don't need to call `/getTopContexts` first unless you want manual control over context selection.

Request Format

```
POST /chat
Content-Type: application/json

{
  "course_name": "CS101",
  "question": "Can you explain polymorphism with a simple example in Python?",
  "conversation_id": "conv-uuid-optional",
  "conversation_history": [
    {
      "role": "user",
      "content": "What is OOP?"
    },
    {
      "role": "assistant",
      "content": "OOP is a programming paradigm based on objects..."
    }
  ]
}
```

Parameters

Parameter	Type	Required	Default	Description
`course_name`	string	✅ Yes	-	Course identifier for context retrieval
`question`	string	✅ Yes	-	User's natural language question
`conversation_id`	string	❌ No	null	For conversation continuity and tracking
`conversation_history`	array[object]	❌ No	[]	Previous messages (last 5 turns used)

Response


```
{
  "answer": "Based on [Source 1 - Lecture_03_OOP_Concepts.pdf] and [Source 2 -

  "contexts": [
    {
      "text": "Polymorphism is one of the four fundamental principles...",
      "readable_filename": "Lecture_03_OOP_Concepts.pdf",
      "score": 0.8947,
      "pagenumber": "12"
    },
    {
      "text": "Example of polymorphism in Python:\n\nclass Animal:...",
      "readable_filename": "Python_Examples.py",
      "score": 0.8201
    }
  ],

  "sources_used": 3,
  "model": "gpt-4.1-mini",

  "usage": {
    "prompt_tokens": 523,
    "completion_tokens": 187,
    "total_tokens": 710
  }
}
```


Processing Flow

1. **Receive Question:** API validates parameters
2. **Context Retrieval (Internal):**
    - Calls getTopContexts internally
    - Retrieves top 5 most relevant chunks
    - Takes ~1.9 seconds
3. **Prompt Construction:**
    - Builds system prompt with course context
    - Adds retrieved context chunks with source citations

- Includes conversation history if provided
- Adds user's current question

4. **AI Generation:**
   - Sends complete prompt to Azure OpenAI (gpt-4.1-mini)
   - Temperature: 0.7 (balanced creativity/accuracy)
   - Max tokens: 1500
   - Takes 2-4 seconds depending on response length
5. **Response Formatting:**
   - Parses AI response
   - Includes source contexts for transparency
   - Adds token usage for monitoring
6. **Background Logging:**
   - Async call to /llm-monitor-message
   - Tracks quality metrics

System Prompt Template

> The following system prompt guides the AI's behavior to ensure educational, accurate responses:

```
You are a helpful AI teaching assistant for the {course_name} course.

Your responsibilities:
1. Answer student questions accurately using ONLY the provided context
2. If the context doesn't contain enough information, say so honestly
3. Cite sources when possible using [Source N - filename] notation
4. Be clear, concise, and educational in your explanations
5. If asked about topics outside course materials, politely redirect

Always prioritize accuracy over making assumptions.
Never invent information not present in the context.
```

Performance Metrics

Stage	Average	P95	Notes
Context Retrieval	1.9s	3.2s	Internal getTopContexts call
Prompt Construction	0.1s	0.2s	Fast in-memory operation
GPT-4.1-mini Generation	2.5s	6.0s	Varies with response length
Response Formatting	0.1s	0.2s	JSON serialization
Total Request Time	**4.6s**	**9.6s**	**Typical student question**

Error Responses

```
// 400 Bad Request
{
  "error": "Missing required parameters",
  "message": "question and course_name are required"
}

// 404 Not Found - No relevant context
{
  "answer": "I don't have enough information in the course materials to answer
  "contexts": [],
  "sources_used": 0
}

// 500 Internal Server Error
{
  "error": "AI generation failed",
  "message": "Azure OpenAI service timeout",
  "code": "GENERATION_ERROR"
}
```

1.4 Streaming Responses

POST /chat/stream

Purpose: Real-time streaming version of /chat for better user experience

Use Cases:

- Show AI "thinking" and typing in real-time
- Reduce perceived latency for long responses
- Better UX for interactive conversations
- Mobile apps with progressive rendering

Request Format

```
POST /chat/stream
Content-Type: application/json

{
  "course_name": "CS101",
  "question": "Explain polymorphism in detail"
}
```

Parameters are identical to /chat but response format is different (streaming vs complete JSON)

Response Format

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Transfer-Encoding: chunked

Based on [Source 1], polymorphism...
[text streams token by token]
...allowing objects to take many forms.
```

Client Implementation Example

```
// JavaScript/TypeScript example
const response = await fetch('/chat/stream', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    course_name: 'CS101',
    question: 'What is polymorphism?'
  })
});

const reader = response.body.getReader();
const decoder = new TextDecoder();

while (true) {
  const { done, value } = await reader.read();
  if (done) break;

  const chunk = decoder.decode(value);
  console.log(chunk); // Display incrementally
}
```

2. Content Ingestion

Beyond manual file uploads, the system supports automated content ingestion from various sources including Learning Management Systems, research databases, and more.

2.1 Canvas LMS Integration

POST /canvas_ingest

Purpose: Bulk import course content directly from Canvas LMS

Use Cases:

- One-click import of entire Canvas course
- Scheduled sync of course materials (nightly updates)
- Import specific content types (files only, no discussions)
- Automated content refresh for large courses

Prerequisites

⚠ **Required:** Canvas Access Token must be configured in environment variables (`CANVAS_ACCESS_TOKEN`). Without this, the endpoint will return 500 error.

Request Format

```
POST /canvas_ingest
Content-Type: application/json

{
  "course_name": "CS101",
  "canvas_url": "https://canvas.illinois.edu/courses/12345",
  "files": true,
  "pages": true,
  "modules": true,
  "syllabus": true,
  "assignments": true,
  "discussions": false
}
```

Parameters

Parameter	Type	Required	Default	Description
`course_name`	string	✅ Yes	-	Your local course identifier
`canvas_url`	string	✅ Yes	-	Full Canvas course URL
`files`	boolean	❌ No	true	Import uploaded files (PDFs, docs, etc.)
`pages`	boolean	❌ No	true	Import wiki pages
`modules`	boolean	❌ No	true	Import module content and structure
`syllabus`	boolean	❌ No	true	Import course syllabus
`assignments`	boolean	❌ No	true	Import assignment descriptions and rubrics
`discussions`	boolean	❌ No	true	Import discussion board threads

Response


```
{
  "outcome": "Queued Canvas Ingest task",
  "ingest_task_ids": [
    "canvas-files-abc123",
    "canvas-pages-def456",
    "canvas-modules-ghi789",
    "canvas-syllabus-jkl012"
  ],
  "success": true,
  "estimated_items": 127
}
```


Processing Flow

1. **Authentication:** Validates Canvas access token
2. **Auto-enrollment:** Accepts pending enrollments for the course
3. **Content Discovery:**
    - Queries Canvas API for each selected content type
    - Counts total items to process
4. **Parallel Task Creation:**
    - Creates separate ingestion task for each content type
    - Each task processed by dedicated worker
    - Up to 6 parallel workers (one per content type)
5. **Worker Processing:**
    - Downloads content from Canvas
    - Converts HTML to text where needed
    - Queues individual file ingestions
    - Progress tracked per task
6. **Completion Notification:**
    - Email sent when all tasks complete
    - Summary of items processed

Typical Processing Time

Course Size	Items	Estimated Time
Small Course	20-50 items	2-5 minutes
Medium Course	50-150 items	5-15 minutes
Large Course	150-500 items	15-45 minutes
Very Large Course	500+ items	45+ minutes

Error Responses

```
// 500 - Missing Canvas token
{
  "error": "Canvas access token not configured",
  "message": "CANVAS_ACCESS_TOKEN environment variable not set"
}

// 500 - Invalid Canvas URL
{
  "error": "Invalid Canvas URL",
  "message": "Could not extract course ID from URL"
}

// 500 - Canvas API error
{
  "error": "Canvas API request failed",
  "message": "Course not found or access denied",
  "canvas_error": "Unauthorized"
}
```

2.2 PubMed Research Integration

GET /pubmedExtraction

Purpose: Import medical research papers from PubMed database

Use Cases:

- Medical school courses with research components
- Clinical knowledge base creation
- Evidence-based medicine support
- Literature review automation

Request Format

```
GET /pubmedExtraction?project_name=MedEd101&search_query=diabetes+treatment+gui
```

Parameters

Parameter	Type	Required	Default	Description
`project_name`	string	✅ Yes	-	Course/project identifier
`search_query`	string	✅ Yes	-	PubMed search terms (use + for spaces)
`count`	integer	❌ No	10	Number of papers to retrieve (1-100)

Response

```
{
  "outcome": "Queued PubMed extraction task",
  "task_id": "pubmed-xyz789",
  "search_query": "diabetes treatment guidelines",
  "papers_found": 20,
  "success": true
}
```

Extracted Content

For each paper, the system attempts to extract:

- **Abstract:** Always available from PubMed
- **Full Text:** If available via PMC (PubMed Central)
- **Metadata:** Authors, journal, publication date, DOI
- **Citations:** References and citing articles
- **MeSH Terms:** Medical Subject Headings for categorization

2.3 Conversation File Upload

POST /process-chat-file

Purpose: Process files uploaded during active conversations for immediate Q&A

Use Cases:

- Student uploads homework for instant AI review
- Quick document analysis during chat
- Instructor shares supplementary material mid-conversation
- Real-time document-based Q&A

> 🚀 **Fast Processing:** Unlike regular ingestion, this endpoint processes synchronously and returns results immediately (10-30 seconds). File is immediately available for questions.

Request Format

```
POST /process-chat-file
Content-Type: application/json

{
  "conversation_id": "conv-uuid-123",
  "s3_path": "uploads/user-456/homework1.pdf",
  "course_name": "CS101",
  "readable_filename": "Homework 1 Submission",
  "user_id": "user-456"
}
```

Parameters

Parameter	Type	Required	Description
`conversation_id`	string	✅ Yes	Active conversation UUID
`s3_path`	string	✅ Yes	Azure Blob path of uploaded file
`course_name`	string	❌ No	Course context (defaults to "chat")
`readable_filename`	string	❌ No	Display name
`user_id`	string	❌ No	User identifier for access control

Response

```
{
  "success": true,
  "chunks_created": 15,
  "status": "completed",
  "message": "File processed and ready for chat"
}
```

Key Differences from Regular Ingestion

Feature	Regular /ingest	/process-chat-file
Processing	Asynchronous	Synchronous
Response Time	200ms (immediate)	10-30s (waits for completion)
Availability	30-60 seconds after upload	Immediately upon response
Scope	Course-wide	Conversation-specific
Use Case	Permanent course materials	Temporary chat context

3. Data Management

Comprehensive tools for viewing, exporting, and managing course documents and conversation data.

3.1 List All Documents

GET /getAll

Purpose: Retrieve complete list of documents for a course

Use Cases:

- Admin dashboard showing course materials
- Document inventory for auditing
- Content management interface
- Duplicate detection

Request Format

```
GET /getAll?course_name=CS101
```

Response


```
{
  "distinct_files": [
    {
      "readable_filename": "Lecture_01_Introduction.pdf",
      "s3_path": "CS101/documents/uuid-abc-123.pdf",
      "url": "https://courses.example.edu/cs101/lecture01.pdf",
      "created_at": "2025-01-15T10:30:00Z",
      "doc_groups": ["lectures", "week1"],
      "page_count": 25,
      "file_size": "2.4 MB",
      "chunk_count": 18
    },
    {
      "readable_filename": "Syllabus_Fall_2025.pdf",
      "s3_path": "CS101/documents/uuid-def-456.pdf",
      "created_at": "2025-01-10T08:00:00Z",
      "doc_groups": ["admin"],
      "page_count": 5
    },
    {
      "readable_filename": "Assignment_01.docx",
      "s3_path": "CS101/assignments/uuid-ghi-789.docx",
      "created_at": "2025-01-20T14:00:00Z",
      "doc_groups": ["assignments"],
      "chunk_count": 8
    }
  ],
  "total_count": 42,
  "total_chunks": 356
}
```

3.2 Delete Documents

DELETE /delete

Purpose: Remove document from all system components

Use Cases:

- Remove outdated course materials
- Delete incorrect uploads
- Content moderation
- GDPR/data retention compliance

⚠ **Warning:** Deletion is asynchronous and "best effort." The endpoint returns success immediately, but actual deletion happens in background. Deleted content may still appear in cached searches for a few minutes.

Request Format

```
DELETE /delete?course_name=CS101&s3_path=CS101/documents/uuid-abc-123.pdf
```

OR identify by URL:

```
DELETE /delete?course_name=CS101&url=https://example.com/old-lecture.pdf
```

Parameters

Parameter	Type	Required	Description
`course_name`	string	✅ Yes	Course identifier
`s3_path`	string	✅*	Azure Blob Storage path of document
`url`	string	✅*	Original source URL of document

* Either `s3_path` or `url` must be provided

Response

```
{
  "outcome": "success",
  "deleted": true,
  "message": "Deletion queued for background processing"
}
```

What Gets Deleted

1. **Azure Blob Storage:** Original file removed
2. **ChromaDB:** All vector embeddings for document chunks
3. **MS SQL Server:** Document metadata and references
4. **Nomic Maps:** Visualization points (on next update)

3.3 Export Document Metadata

GET /exportDocuments

Purpose: Export complete document inventory as JSON or CSV

Use Cases:

- Backup document registry
- Content audit reports
- Data migration preparation
- Analytics and reporting

Request Format

```
GET /exportDocuments?course_name=CS101&format=csv
```

Parameters

Parameter	Type	Default	Description
`course_name`	string	required	Course to export
`format`	string	json	Output format: json or csv

Response (CSV)

```
filename,s3_path,upload_date,page_count,chunk_count,doc_groups,file_size
"Lecture_01.pdf","CS101/uuid-123.pdf","2025-01-15T10:30:00Z",25,18,"lectures|we
"Syllabus.pdf","CS101/uuid-456.pdf","2025-01-10T08:00:00Z",5,7,"admin","340 KB"
```

3.4 Export Conversation Data

GET /export-convo-history

Purpose: Export conversation history for analysis, backup, or training

Use Cases:

- Research on student learning patterns
- AI model fine-tuning data collection
- Quality assurance and monitoring
- Student engagement analytics

Request Format

```
GET /export-convo-history?course_name=CS101&format=json&from_date=2025-01-01&to
```

Parameters

Parameter	Type	Default	Description
`course_name`	string	required	Course filter
`format`	string	json	json or csv
`from_date`	string	null	ISO date (YYYY-MM-DD)
`to_date`	string	null	ISO date (YYYY-MM-DD)
`user_id`	string	null	Filter by specific user

Response (JSON)


```
{
  "conversations": [
    {
      "id": "conv-uuid-123",
      "course_name": "CS101",
      "user_id": "user-456",
      "created_at": "2025-01-15T14:23:00Z",
      "messages": [
        {
          "role": "user",
          "content": "What is polymorphism?",
          "timestamp": "2025-01-15T14:23:00Z"
        },
        {
          "role": "assistant",
          "content": "Based on [Source 1], polymorphism is...",
          "timestamp": "2025-01-15T14:23:05Z",
          "contexts_used": 3,
          "tokens": 245,
          "model": "gpt-4.1-mini"
        }
      ],
      "total_messages": 6,
      "satisfaction_rating": 4.5,
      "duration_seconds": 420
    }
  ],
  "total_conversations": 156,
  "total_messages": 892,
  "date_range": {
    "from": "2025-01-01",
    "to": "2025-01-31"
  },
  "exported_at": "2025-02-01T10:00:00Z"
}
```

Privacy Considerations

🔒 **Privacy:** Exported data may contain student questions and interactions. Ensure FERPA compliance:

- Anonymize user identifiers if sharing externally
- Restrict access to authorized personnel only
- Store securely with encryption at rest
- Delete after retention period expires

4. Analytics & Monitoring

Comprehensive metrics and insights for course performance, user engagement, and system health.

4.1 Project Statistics

GET /getProjectStats

Purpose: Get high-level overview of course metrics

Use Cases:

- Admin dashboard summary
- Course health monitoring
- Resource usage tracking
- ROI analysis

Request Format

```
GET /getProjectStats?course_name=CS101
```

Response

```
{
  "course_name": "CS101",
  "created_at": "2024-09-01T00:00:00Z",
  "last_activity": "2025-01-20T14:30:00Z",

  "documents": {
    "total": 45,
    "by_type": {
      "pdf": 32,
      "video": 8,
      "docx": 3,
      "other": 2
    },
    "total_chunks": 456,
    "total_size_mb": 234.5
  },

  "conversations": {
    "total": 1250,
    "this_week": 87,
    "this_month": 342,
    "avg_per_day": 12.4,
    "avg_messages_per_conversation": 4.2
  },

  "users": {
    "total_unique": 234,
    "active_this_week": 89,
    "active_this_month": 178
  },

  "ai_usage": {
    "total_tokens": 5234567,
    "this_month_tokens": 234567,
    "estimated_cost_usd": 52.35,
    "avg_response_time_seconds": 4.6
  },

  "engagement": {
    "avg_satisfaction": 4.3,
```

```
    "questions_answered": 5234,
    "unique_topics": 342
  }
}
```

4.2 Weekly Trends

GET /getWeeklyTrends

Purpose: Track usage patterns and trends over time

Request Format

```
GET /getWeeklyTrends?course_name=CS101&weeks=8
```

Response


```
{
  "course_name": "CS101",
  "weeks": [
    {
      "week_start": "2025-01-13",
      "week_end": "2025-01-19",
      "conversations": 87,
      "unique_users": 45,
      "messages": 358,
      "avg_response_time": 4.3,
      "satisfaction_rate": 4.2,
      "peak_hour": 14,
      "total_tokens": 45678
    },
    {
      "week_start": "2025-01-06",
      "conversations": 92,
      "unique_users": 48,
      "satisfaction_rate": 4.5
    }
  ],
  "trend_analysis": {
    "conversations_trend": "increasing",
    "satisfaction_trend": "stable",
    "peak_usage_day": "Tuesday",
    "peak_usage_hour": 14
  }
}
```

4.3 AI Model Usage

GET /getModelUsageCounts

Purpose: Track AI model usage and costs

Response


```
{
  "period": {
    "start": "2025-01-01",
    "end": "2025-01-31"
  },
  "models": [
    {
      "model": "gpt-4.1-mini",
      "requests": 1234,
      "tokens": {
        "prompt": 1567890,
        "completion": 789012,
        "total": 2356902
      },
      "avg_tokens_per_request": 1910,
      "estimated_cost_usd": 23.57
    },
    {
      "model": "text-embedding-3-small",
      "requests": 5678,
      "tokens": 8901234,
      "estimated_cost_usd": 0.89
    }
  ],
  "total_cost_usd": 24.46,
  "cost_breakdown": {
    "chat_generation": 23.57,
    "embeddings": 0.89
  }
}
```

4.4 Conversation Analytics

GET /getConversationStats

Purpose: Deep dive into conversation patterns

Response

```
{
  "course_name": "CS101",
  "period": "last_30_days",

  "summary": {
    "total_conversations": 1250,
    "total_messages": 5234,
    "avg_messages_per_conversation": 4.2,
    "avg_conversation_duration_seconds": 420
  },

  "satisfaction": {
    "average_rating": 4.3,
    "ratings_distribution": {
      "5_stars": 45,
      "4_stars": 35,
      "3_stars": 15,
      "2_stars": 3,
      "1_star": 2
    },
    "with_feedback_percent": 23
  },

  "top_topics": [
    {"topic": "polymorphism", "count": 45, "avg_satisfaction": 4.5},
    {"topic": "inheritance", "count": 38, "avg_satisfaction": 4.3},
    {"topic": "data structures", "count": 32, "avg_satisfaction": 4.1}
  ],

  "temporal_patterns": {
    "peak_hours": [14, 15, 16, 20],
    "peak_days": ["Tuesday", "Wednesday", "Thursday"],
    "busiest_hour": "2pm-3pm",
    "slowest_hour": "3am-4am"
  },

  "performance": {
    "avg_response_time_seconds": 4.6,
    "response_time_p50": 3.9,
    "response_time_p95": 8.2,
```

```
    "response_time_p99": 12.1
  },

  "quality_metrics": {
    "context_relevance_avg": 0.87,
    "answer_accuracy_estimated": 0.91,
    "source_citation_rate": 0.95
  }
}
```

4.5 Conversation Monitoring

POST /llm-monitor-message

Purpose: Track individual conversation quality and metrics

Use Cases:

- Real-time quality monitoring
- Detect problematic responses
- User satisfaction tracking
- Model performance analysis

Automatic: This endpoint is called automatically after each /chat request. Manual calls are optional for custom tracking.

Request Format

```
POST /llm-monitor-message
Content-Type: application/json

{
  "course_name": "CS101",
  "conversation_id": "conv-uuid-123",
  "user_email": "student@university.edu",
  "model_name": "gpt-4.1-mini"
}
```

Response

```
{
  "outcome": "Task started",
  "monitoring_enabled": true
}
```

What Gets Tracked

- **Response Quality:** Context relevance, answer coherence
- **Performance:** Response time, token usage
- **User Behavior:** Follow-up questions, satisfaction signals
- **Error Rates:** Failed retrievals, generation errors
- **Cost Attribution:** Per-user, per-course token usage

5. Visualization

Interactive document and conversation maps using Nomic Atlas for data exploration and insights.

5.1 View Nomic Maps

GET /getNomicMap

Purpose: Get interactive 2D visualization of documents or conversations

Use Cases:

- Explore document clustering and topics
- Identify knowledge gaps in course materials
- Visualize conversation themes
- Discover content relationships

Request Format

```
GET /getNomicMap?course_name=CS101&map_type=document
```

Parameters

Parameter	Type	Options	Description
`course_name`	string	-	Course identifier
`map_type`	string	document, conversation	Type of visualization

Response


```
{
  "map_url": "https://atlas.nomic.ai/map/abc-123-def-456",
  "embed_html": "<iframe src='https://atlas.nomic.ai/map/abc-123/embed' width='
  "map_id": "abc-123-def-456",
  "total_points": 450,
  "last_updated": "2025-01-20T10:00:00Z",

  "clusters": [
    {
      "id": 1,
      "label": "OOP Concepts",
      "size": 125,
      "centroid": [0.234, -0.567]
    },
    {
      "id": 2,
      "label": "Data Structures",
      "size": 98,
      "centroid": [-0.123, 0.456]
    },
    {
      "id": 3,
      "label": "Algorithms",
      "size": 87
    }
  ],

  "insights": {
    "most_central_topic": "Object-Oriented Programming",
    "isolated_topics": ["Advanced Concurrency"],
    "largest_cluster": "OOP Concepts"
  }
}
```

Embedding in Web Pages

```
<!-- Embed Nomic map in your web app -->
<iframe
  src="https://atlas.nomic.ai/map/abc-123-def-456/embed"
  width="100%"
  height="600"
  frameborder="0"
  style="border-radius: 8px;"
></iframe>
```

5.2 Create Document Map

GET /createDocumentMap

Purpose: Initialize new document visualization for a course

Request Format

```
GET /createDocumentMap?course_name=CS101
```

Processing

1. Fetches all document embeddings from ChromaDB
2. Applies dimensionality reduction (UMAP)
3. Creates interactive atlas on Nomic platform
4. Returns map URL and embed code
5. Typical time: 2-5 minutes for 500 documents

5.3 Create Conversation Map

GET /createConversationMap

Purpose: Initialize conversation topic visualization

Similar to document map but visualizes conversation themes and user questions.

5.4 Map Update Operations

GET /updateDocumentMaps

Purpose: Refresh document map with newly added materials

```
GET /updateDocumentMaps?course_name=CS101
```

GET /updateConversationMaps

Purpose: Refresh conversation map with recent discussions

GET /cleanUpDocumentMaps

Purpose: Remove deleted documents from visualization

GET /cleanUpConversationMaps

Purpose: Remove old/irrelevant conversations

Automated: Maps are automatically updated daily via cron job at 6:00 AM UTC. Manual updates only needed for immediate refresh.

6. Advanced Features

Enhanced retrieval techniques, custom workflows, and knowledge graph integration.

6.1 Workflow Management

GET /getworkflows

> **Purpose:** List available RAG workflows

Response

```
{
  "workflows": [
    {
      "id": "default-rag",
      "name": "Standard RAG",
      "description": "Simple retrieve-then-generate",
      "active": true
    },
    {
      "id": "advanced-mqr",
      "name": "Multi-Query Retrieval",
      "description": "Generate multiple query variations",
      "active": false
    },
    {
      "id": "kg-enhanced",
      "name": "Knowledge Graph Enhanced",
      "description": "Combine vector search with knowledge graphs",
      "active": false
    }
  ]
}
```

GET /switch_workflow

Purpose: Change active workflow

```
GET /switch_workflow?course_name=CS101&workflow_id=advanced-mqr
```

POST /run_flow

Purpose: Execute custom workflow

```
POST /run_flow
{
  "course_name": "CS101",
  "workflow_id": "advanced-mqr",
  "input": {"question": "What is polymorphism?"}
}
```

6.2 Multi-Query Retrieval

GET /getTopContextsWithMQR

Purpose: Enhanced retrieval using multiple query variations

How It Works

1. Original query: "What is polymorphism?"
2. Generates variations:
    - "Explain polymorphism in OOP"
    - "Define polymorphism concept"
    - "Polymorphism examples and uses"
3. Searches with each variation
4. Merges and deduplicates results
5. Reranks by relevance
6. Returns top N unified results

Benefits

- **Higher Recall:** Catches relevant docs missed by single query
- **Better for Ambiguous Questions:** Multiple perspectives
- **Robust to Query Formulation:** Less sensitive to exact wording

Trade-offs

- **Slower:** 3-5x processing time (multiple searches)

- **More Expensive:** Multiple embedding API calls
- **Use Selectively:** Best for complex research queries

6.3 Knowledge Graph Queries

GET /getClinicalKGContexts

> **Purpose:** Query clinical knowledge graph for medical courses

Request

```
GET /getClinicalKGContexts?entity=diabetes&relation=treats
```

Use Cases

- Medical education courses
- Drug interaction checking
- Clinical decision support
- Evidence-based medicine

GET /getPrimeKGContexts

Purpose: Query PrimeKG biomedical knowledge graph

Contains structured data on:

- Diseases and conditions
- Drugs and treatments
- Genes and proteins
- Biological pathways
- Clinical relationships

7. Utilities

Administrative and maintenance endpoints for system management.

7.1 Create Project

POST /createProject

Purpose: Initialize new course/project in the system

Use Cases:

- Set up new course for semester
- Create research project workspace
- Initialize department knowledge base

Request Format

```
POST /createProject
Content-Type: application/json

{
  "course_name": "CS101",
  "course_title": "Introduction to Computer Science",
  "owner_id": "prof-123",
  "settings": {
    "public": false,
    "allow_student_uploads": true,
    "enable_analytics": true
  }
}
```

Parameters

Parameter	Type	Required	Description
`course_name`	string	✅ Yes	Unique identifier (no spaces or special chars)
`course_title`	string	❌ No	Human-readable course name
`owner_id`	string	❌ No	Instructor/admin user ID
`settings`	object	❌ No	Course configuration

Response

```
{
  "project_id": "proj-abc-123",
  "course_name": "CS101",
  "created_at": "2025-01-20T10:00:00Z",
  "success": true,
  "resources_created": {
    "database_entry": true,
    "azure_blob_container": true,
    "chromadb_collection": true,
    "nomic_maps": "queued"
  }
}
```

What Gets Created

1. **Database Entry:** Project metadata in MS SQL Server
2. **Storage Container:** Azure Blob container for files
3. **Vector Collection:** ChromaDB collection for embeddings
4. **Visualizations:** Nomic map spaces (created async)
5. **Default Settings:** Access controls, quotas, preferences

7.2 Update Project Documents

GET /updateProjectDocuments

Purpose: Synchronize document registry with actual storage

Use Cases:

- After bulk file uploads to Azure Blob
- Reconcile after manual changes
- Audit document inventory
- Fix metadata inconsistencies

Request

```
GET /updateProjectDocuments?course_name=CS101
```

Processing

1. Scans Azure Blob Storage for files
2. Queries ChromaDB for vectors
3. Compares with MS SQL metadata
4. Reconciles differences:
    - Adds missing database entries
    - Removes orphaned records
    - Updates file sizes and dates

Response

```
{
  "success": true,
  "documents_found": 45,
  "database_updated": 3,
  "orphaned_removed": 1,
  "discrepancies": []
}
```

7.3 Send Transactional Email

POST /send-transactional-email

Purpose: Send automated emails to users

Request Format

```
POST /send-transactional-email
Content-Type: application/json

{
  "to_email": "student@university.edu",
  "subject": "Your document has been processed",
  "template": "ingestion_complete",
  "data": {
    "course_name": "CS101",
    "document_name": "Lecture 01.pdf",
    "chunks_created": 25,
    "processing_time": "45 seconds"
  }
}
```

Available Templates

- `ingestion_complete` - Document processing finished
- `ingestion_failed` - Processing error notification
- `welcome` - New course enrollment
- `weekly_digest` - Usage summary
- `export_ready` - Data export completed

7.4 System Health Check

GET /health

Purpose: Verify system components are operational

Request

```
GET /health
```

Response

```
{
  "status": "healthy",
  "service": "ai-ta-backend",
  "timestamp": 1705745432,
  "version": "1.0.0",

  "components": {
    "api": {
      "status": "ok",
      "response_time_ms": 5
    },
    "database": {
      "status": "ok",
      "connection_pool": "8/20 active"
    },
    "vector_db": {
      "status": "ok",
      "collection_count": 15,
      "total_documents": 12456
    },
    "storage": {
      "status": "ok",
      "used_gb": 234.5
    },
    "queue": {
      "status": "ok",
      "pending_tasks": 3
    },
    "ai_services": {
      "status": "ok",
      "embeddings": "operational",
      "chat": "operational"
    }
  },

  "recent_errors": 0,
  "uptime_seconds": 3456789
}
```

Status Codes

HTTP Status	Meaning	Action
200	All systems operational	None
503	One or more components degraded	Check component details
500	Critical system failure	Alert operations team