Database and RAG

Canvas Database and RAG System Documentation

Overview

A system for storing and retrieving Canvas LMS content using both SQL and vector databases, with RAG (Retrieval Augmented Generation) capabilities.

Database System (db.py)

Class: CanvasDatabase

Properties

- `sqlite_conn`: SQLite database connection
- `cursor`: SQLite cursor for executing commands
- `chroma_client`: ChromaDB client for vector storage
- `collection`: ChromaDB collection for storing vector representations

Core Methods

`__init__()`

- Initializes SQLite connection
- Sets up ChromaDB client
- Creates collection named "canvas_content"
- Calls init_tables()

`init_tables()`

Creates the following SQL tables with appropriate fields and relationships:

- courses

- assignments
- modules
- module_items
- announcements
- discussion_posts
- files
- pages
- submissions
- calendar_events
- grades

Also creates necessary indices for performance optimization.

`store_canvas_item(item_type: str, data: Dict[str, Any]) -> bool`

Stores Canvas items in both databases:

1. Checks if item exists
2. Stores in SQLite
3. Prepares content for ChromaDB
4. Adds to vector database
5. Returns success status

`store_file(course_id: str, file_name: str, file_data: bytes, content_type: str) -> Optional[str]`

Specialized method for file storage:

1. Generates unique file ID
2. Extracts text from file based on type
3. Stores in SQLite with binary data
4. Adds to ChromaDB if text was extracted

5. Returns file ID or None on failure

`query_content(query_text: str, n_results: int = 5) -> List[Dict[str, Any]]`

Performs similarity search:

1. Queries ChromaDB
2. Retrieves additional details from SQLite
3. Returns formatted results with content, metadata, and details

Helper Methods

`_item_exists(item_type: str, item_id: str) -> bool`

- Checks if item exists in appropriate table
- Maps item types to table names
- Returns boolean

`_prepare_content(item_type: str, data: Dict[str, Any]) -> str`

Formats content for ChromaDB based on item type:

- Courses: name, description, syllabus
- Assignments: title, description, due date
- Announcements: title, message
- Discussions: title, message
- Pages: title, body

`_store_in_sqlite(item_type: str, data: Dict[str, Any])`

Handles SQLite storage:

1. Maps item type to table
2. Gets column names
3. Filters data to match columns
4. Generates and executes SQL

`_extract_file_text(file_data: bytes, content_type: str) -> str`

Extracts text from various file types:

- PDF (using PyPDF2)
- Word (using python-docx)
- Plain text
- Markdown
- HTML (using BeautifulSoup)
- Images (metadata only)

RAG System (rag.py)

Class: CanvasRAG

Properties

- `db`: Instance of CanvasDatabase

Core Methods

`__init__()`

- Initializes CanvasDatabase instance

`query_llm(user_query: str) -> Dict[str, Any]`

Main RAG pipeline:

1. Retrieves relevant context from database
2. Formats context for LLM
3. Creates prompt
4. Gets LLM response
5. Returns answer and sources

Helper Methods

`_format_context_for_llm(context: List[Dict[str, Any]]) -> str`

Formats different content types for LLM consumption:

- Assignments: title, due date, description, points
- Files: filename, content
- Courses: general information
- Announcements: title, message
- Pages: title, body
- Modules: name, position

`_create_llm_prompt(user_query: str, context: str) -> str`

Creates structured prompt with:

1. System context
2. Relevant course content
3. User question
4. Response instructions

`_get_llm_response(prompt: str) -> str`

Placeholder for LLM integration

Data Flow

1. **Content Storage Flow**

   ```
   Canvas Item → CanvasDatabase.store_canvas_item()
   ↓
   Check Existence → SQLite Storage → Content Preparation → C
   hromaDB Storage
   ```

2. **File Storage Flow**

   ```
   File → CanvasDatabase.store_file()
   ↓
   ```

```
Generate ID → Extract Text → SQLite Storage → ChromaDB Sto
rage (if text)
```

3. **Query Flow**

```
User Query → CanvasRAG.query_llm()
↓
Database Query → Context Formatting → Prompt Creation → LL
M Response
```

Dependencies

- sqlite3: SQL database operations
- chromadb: Vector database operations
- PyPDF2: PDF text extraction
- python-docx: Word document processing
- BeautifulSoup4: HTML parsing
- datetime: Timestamp management
- typing: Type hints

Notes on Implementation

- Uses both traditional SQL and vector databases for efficient storage and retrieval
- Implements comprehensive error handling throughout
- Modular design allows for easy extension
- Prepared for multi-format content handling
- Ready for LLM integration
- Includes performance optimization through indexing

```
# CanvasDatabase Methods Layout

1. INITIALIZATION METHODS
   ├── __init__(self)
   │   ├── Purpose: Initializes database connections
   │   ├── Sets up: SQLite and ChromaDB connections
   │   └── Called: When creating a new CanvasDatabase instance
   │
   └── init_tables(self)
       ├── Purpose: Creates all database tables
       ├── Creates: All Canvas-related tables (courses, assignme
       └── Called: During initialization

2. MAIN PUBLIC METHODS
   ├── store_canvas_item(self, item_type: str, data: Dict[str, A
   │   ├── Purpose: Main method for storing Canvas content
   │   ├── Handles: Courses, assignments, announcements, etc.
   │   └── Uses: _store_in_sqlite(), _prepare_content()
   │
   ├── store_file(self, course_id: str, file_name: str, file_dat
   │   ├── Purpose: Stores files and their extracted text
   │   ├── Handles: PDFs, Word docs, text files, etc.
   │   └── Uses: _extract_file_text()
   │
   ├── query_content(self, query_text: str, n_results: int = 5)
   │   ├── Purpose: Searches across all content
   │   ├── Uses: ChromaDB for similarity search
   │   └── Uses: _get_item_details()
   │
   └── close(self)
       ├── Purpose: Closes database connections
       └── Called: When finishing database operations

3. PRIVATE HELPER METHODS
   ├── _prepare_content(self, item_type: str, data: Dict[str, Ar
   │   ├── Purpose: Formats content for ChromaDB storage
   │   └── Called by: store_canvas_item()
   │
   ├── _store_in_sqlite(self, item_type: str, data: Dict[str, Ar
   │   ├── Purpose: Handles SQLite storage logic
   │   └── Called by: store_canvas_item()
   │
   ├── _get_item_details(self, item_type: str, item_id: str) ->
   │   ├── Purpose: Retrieves detailed item information
   │   └── Called by: query_content()
   │
   └── _extract_file_text(self, file_data: bytes, content_type:
       ├── Purpose: Extracts text from files
       └── Called by: store_file()

4. DATA FLOW
   ├── Input Data Flow
   │   ├── Canvas content → store_canvas_item() → SQLite + Chron
   │   └── Files → store_file() → SQLite + ChromaDB
   │
   └── Output Data Flow
       └── Query → query_content() → ChromaDB search → SQLite de

5. DATABASE INTERACTIONS
   ├── SQLite (Structured Data)
   │   ├── Stores: Metadata, relationships, structured content
   │   └── Used by: _store_in_sqlite(), _get_item_details()
   │
   └── ChromaDB (Vector Database)
       ├── Stores: Text content, vector embeddings
       └── Used by: store_canvas_item(), query_content()

6. MAIN USE CASES
   ├── Storing Course Content
   │   db.store_canvas_item('course', course_data)
   │
   ├── Storing Files
   │   db.store_file(course_id, file_name, file_data, content_ty
   │
   └── Querying Content
       results = db.query_content("search term")
```