High Level Design Document

For

Multimodal Education Creator

BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING

BY

Sr.no	Student Name	Enrolment Number
1	ABHIKET THAKUR	EN22CS301030
2	ADITYA PALIWAL	EN22CS301055
3	AARUSHI MITTAL	EN24CS3T10026
4	AKASH KUSHWAHA	EN22CS301083

Datagami Skill Based Course

Group: 04D1

Project No: GAI-04

Department of Computer Science & Engineering
Faculty of Engineering
MEDICAPS UNIVERSITY, INDORE- 453331
Jan - May 2026

1. Introduction

1.1 Scope of the Document

This document presents the High-Level Design (HLD) of the AI-Powered Educational Content Creator System. It provides a comprehensive overview of the system architecture, major components, data flow, technology stack, and deployment structure.

The purpose of this document is to**:**

- Define the architectural design of the system.
- Describe the interaction between major modules.
- Provide technical clarity for evaluators.
- Serve as a reference for development, maintenance, and future enhancements

1.2 Intended Audience

The primary audience includes**:**

- Project Evaluators and Academic Reviewers
- Development Team Members
- Future Maintainers and Enhancers
- Research-Oriented Audience

1.3 System overview

The AI-Based Educational Content Creator is an intelligent web-based system designed to generate structured and context-aware educational material using Natural Language Processing (NLP) and semantic vector retrieval techniques. The system leverages a Retrieval-Augmented Generation (RAG) architecture to enhance the relevance and accuracy of generated educational content by retrieving semantically similar knowledge from a pre-indexed vector database.

Purpose of the System: The primary objective of the system is to,

- Automate the generation of structured educational content
- Reduce manual effort in content creation
- Improve contextual relevance using semantic search
- Demonstrate practical implementation of NLP and vector-based retrieval

Core Functionality: The system performs the following high-level operations,

- Accepts a user-provided topic or prompt through a web interface
- Preprocesses and converts the input into a semantic embedding
- Performs similarity search on a precomputed vector database
- Retrieves relevant educational content chunks
- Constructs structured and formatted educational material

Architectural Approach: The system follows a layered modular architecture, consisting of,

- Presentation Layer (Web Interface)
- Application & NLP Processing Layer
- Vector Retrieval Layer
- Knowledge Storage Layer

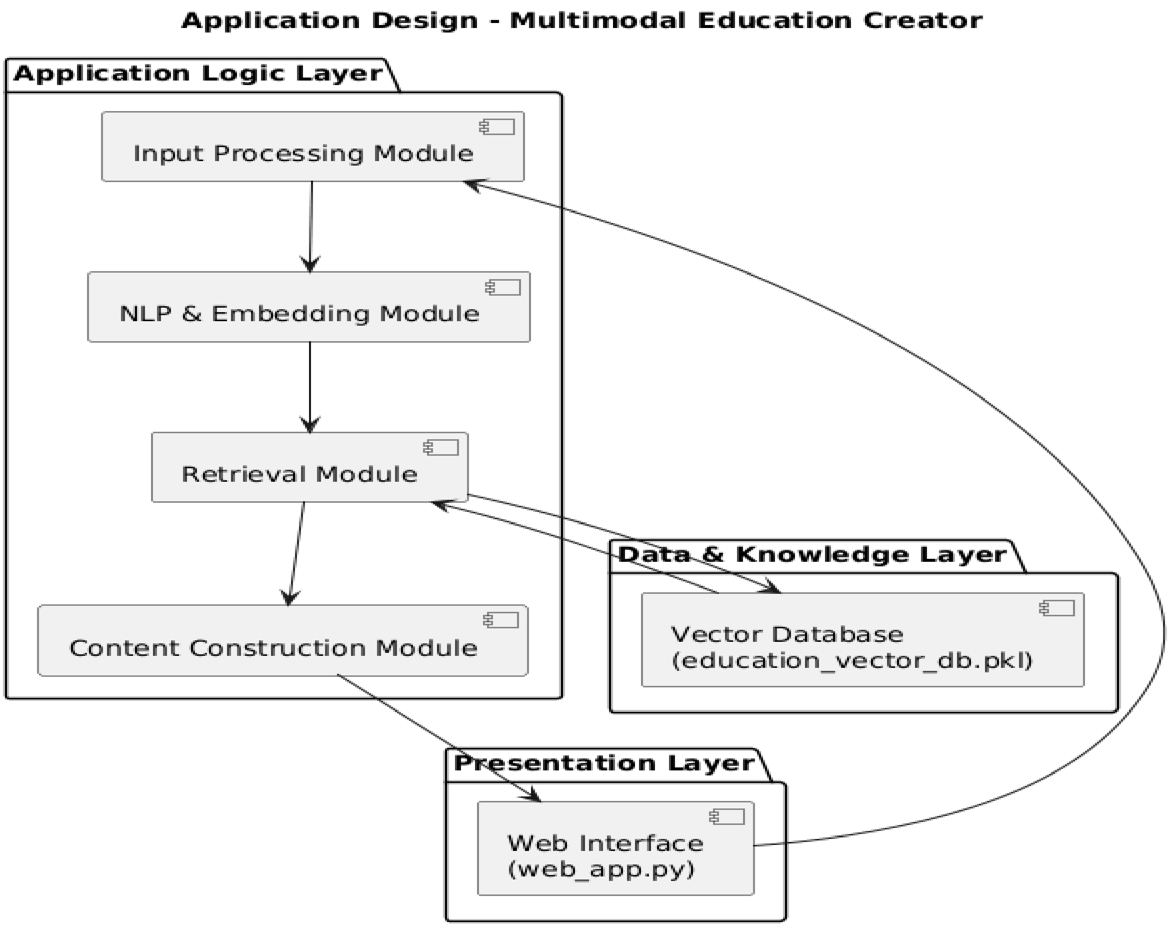
Design Philosophy: The system is designed with,

- **Modularity** – Each component can be independently upgraded
- **Performance Optimization** – Precomputed embeddings reduce runtime cost
- **Scalability Readiness** – Architecture supports migration to distributed vector databases
- **Minimal Data Retention** – No permanent storage of user data

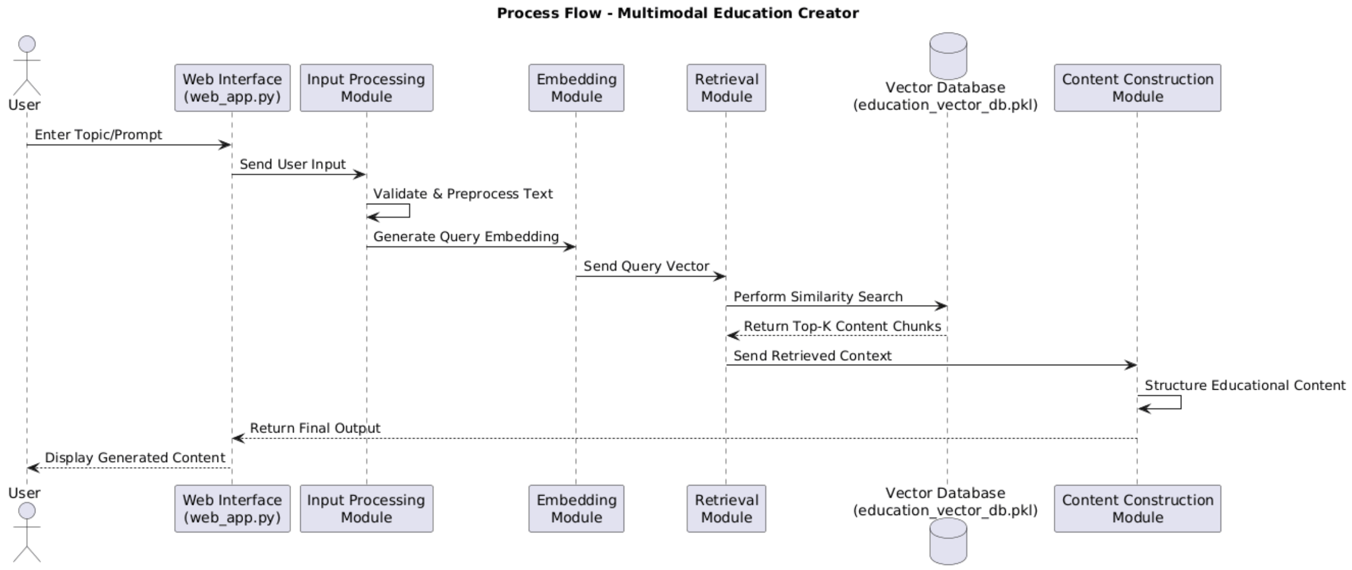
2. System Design

2.1 Application Design

The application follows a layered modular architecture implementing a Retrieval-Augmented Generation (RAG) workflow. It separates user interaction, processing logic, and data storage to ensure maintainability and scalability.



2.2 Process Flow



The system converts the user query into embeddings, retrieves semantically relevant content from the vector database, and generates structured educational material using a RAG-based workflow.

- **User Input**: User enters a topic or prompt through the web interface.
- **Input Validation & Preprocessing**: Text is cleaned, normalized, and prepared for embedding generation.
- **Embedding Generation**: The processed query is converted into a semantic vector representation.
- **Vector Similarity Search**: Query embedding is compared with stored content embeddings using cosine similarity to retrieve Top-K relevant chunks.
- **Context Assembly**: Retrieved content chunks are combined to form contextual knowledge.
- **Content Generation**: Structured educational material is generated using retrieved context.
- **Response Display**: Final formatted output is returned and displayed to the user.

2.3 Information Flow

Information flows through the system in a structured, sequential manner — transforming raw user input into semantically enriched educational content.

Information Movement Across Layers:

User Layer → Presentation Layer

- User submits topic or prompt
- Request is captured by the web interface

Presentation Layer → Processing Layer

- Input is validated and preprocessed
- Cleaned text is forwarded for embedding generation

Processing Layer → Data Layer

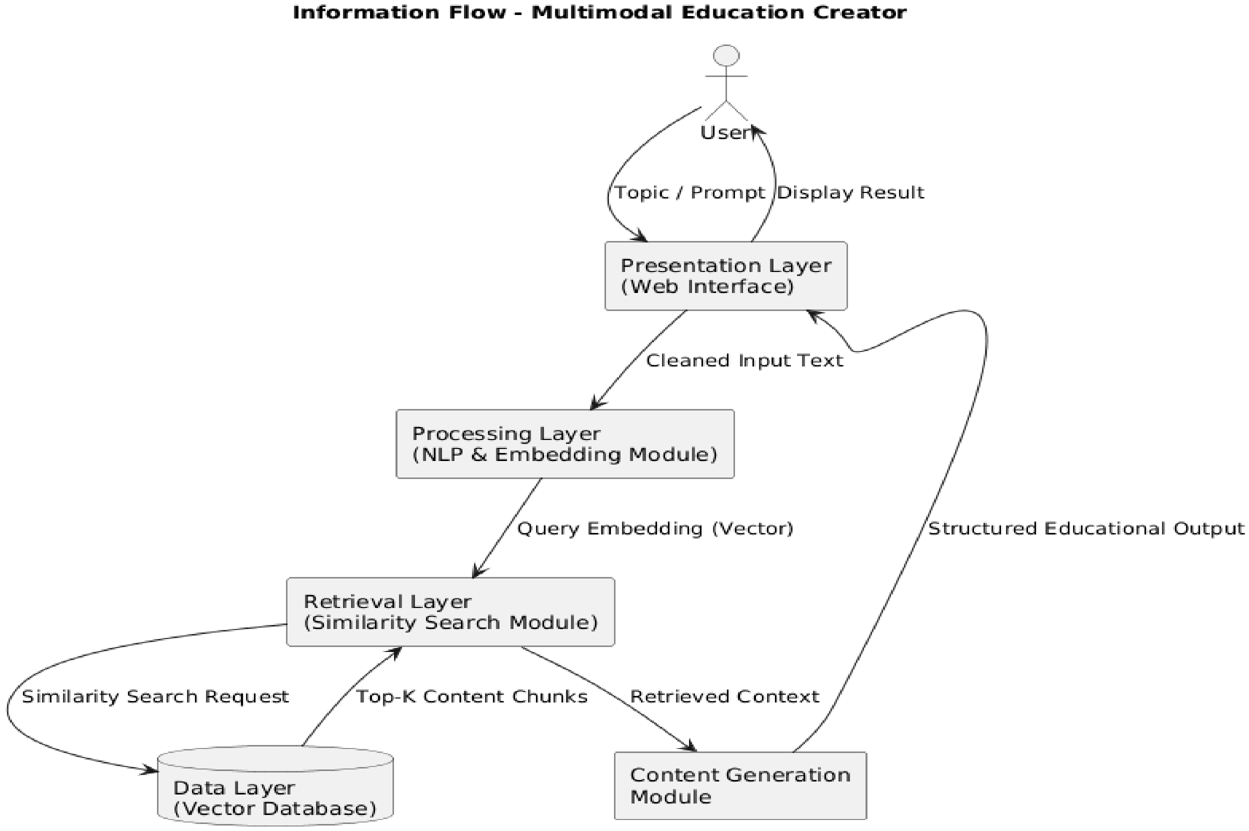
- Query is converted into embedding vector
- Vector is compared with stored content embeddings
- Top-K relevant content chunks are retrieved

Data Layer → Processing Layer

- Retrieved contextual information is assembled
- Structured educational content is generated

Processing Layer → Presentation Layer

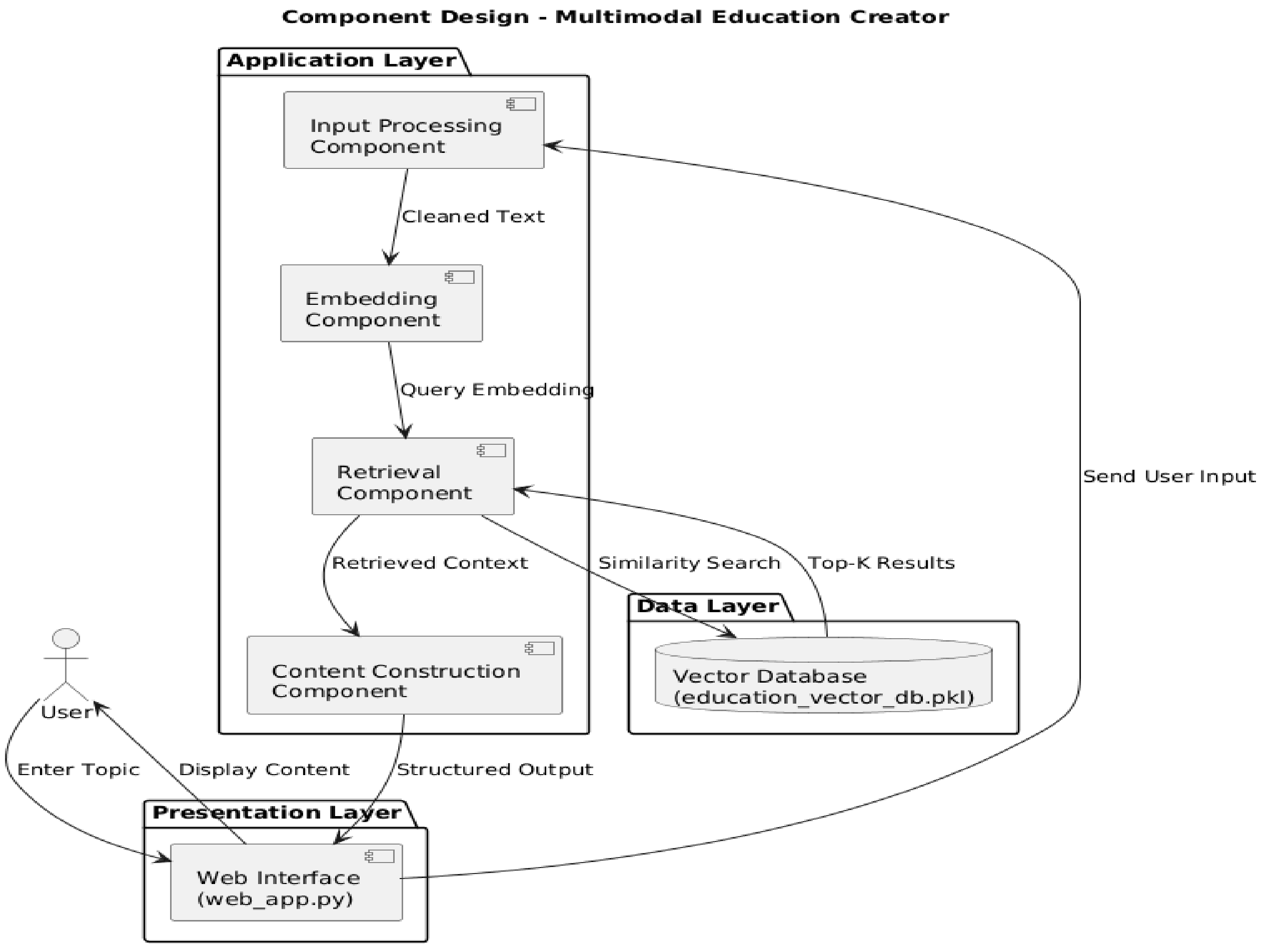
- Final formatted output is returned
- Content is displayed to the user



2.4 Components Design

The system is designed using modular components, where each component has a clearly defined responsibility. This ensures maintainability, scalability, and independent upgrades.

- **Web Interface Component:** Captures user topic or prompt and Sends request to backend. Displays structured educational output and Maintains request-response lifecycle.
- **Input Processing Component:** Validates and sanitizes user input and Performs text cleaning and normalization. Prepares input for embedding generation.
- **Embedding Component:** Converts processed text into semantic vector and Enables similarity-based retrieval. Represents contextual meaning in high-dimensional space.
- **Retrieval Component:** Performs cosine similarity computation and Filters based on similarity score. Retrieves Top-K relevant content chunks.
- **Content Construction Component:** Combines retrieved context and Formats headings and explanations. Structures content into organized educational format.
- **Vector Database Component:** Stores precomputed embeddings and supports fast semantic search and stores content chunks and metadata.



2.5 Key Design Considerations

1. Modularity: The system is divided into independent components (UI, NLP, Retrieval, Vector DB) to ensure maintainability and easy upgrades.

2. Performance Optimization

- Precomputed embeddings reduce runtime computation.
- In-memory vector loading minimizes latency.

- Top-K retrieval limits unnecessary processing.

3. Scalability

- Architecture supports migration to distributed vector databases (e.g., FAISS).
- Can integrate caching and load balancing for high traffic.
- Suitable for containerized cloud deployment.

4. Accuracy & Relevance

- Uses semantic embeddings instead of keyword matching.
- Implements Retrieval-Augmented Generation (RAG) for context-aware output.
- Reduces hallucination by grounding responses in stored knowledge.

5. Security & Data Privacy

- Input validation and sanitization.
- Controlled access to serialized vector database.
- No permanent storage of user prompts.

6. Maintainability

- Clear separation of concerns.
- Replaceable embedding model or storage system.
- Structured codebase with defined module responsibilities.

7. Stateless Processing

- Each query processed independently.
- No cross-session data sharing.
- Simplifies scaling and deployment.

2.6 API Catalogue

The system exposes internal application APIs to handle user requests, process queries, perform semantic retrieval, and generate structured educational content.

I. **Generate Educational Content:** Accepts a user topic or prompt and returns structured educational material using RAG-based retrieval.

  - Endpoint: ***POST /generate-content***
  - Request Body (JSON): {"topic": "Machine Learning"}
  - Response (JSON): {"status": "success", "generated_content": "Structured educational explanation..."}

II. **Health Check API:** Verifies that the application and vector database are loaded and running.

- Endpoint: **GET /health**
- Response: {"status": "active"}

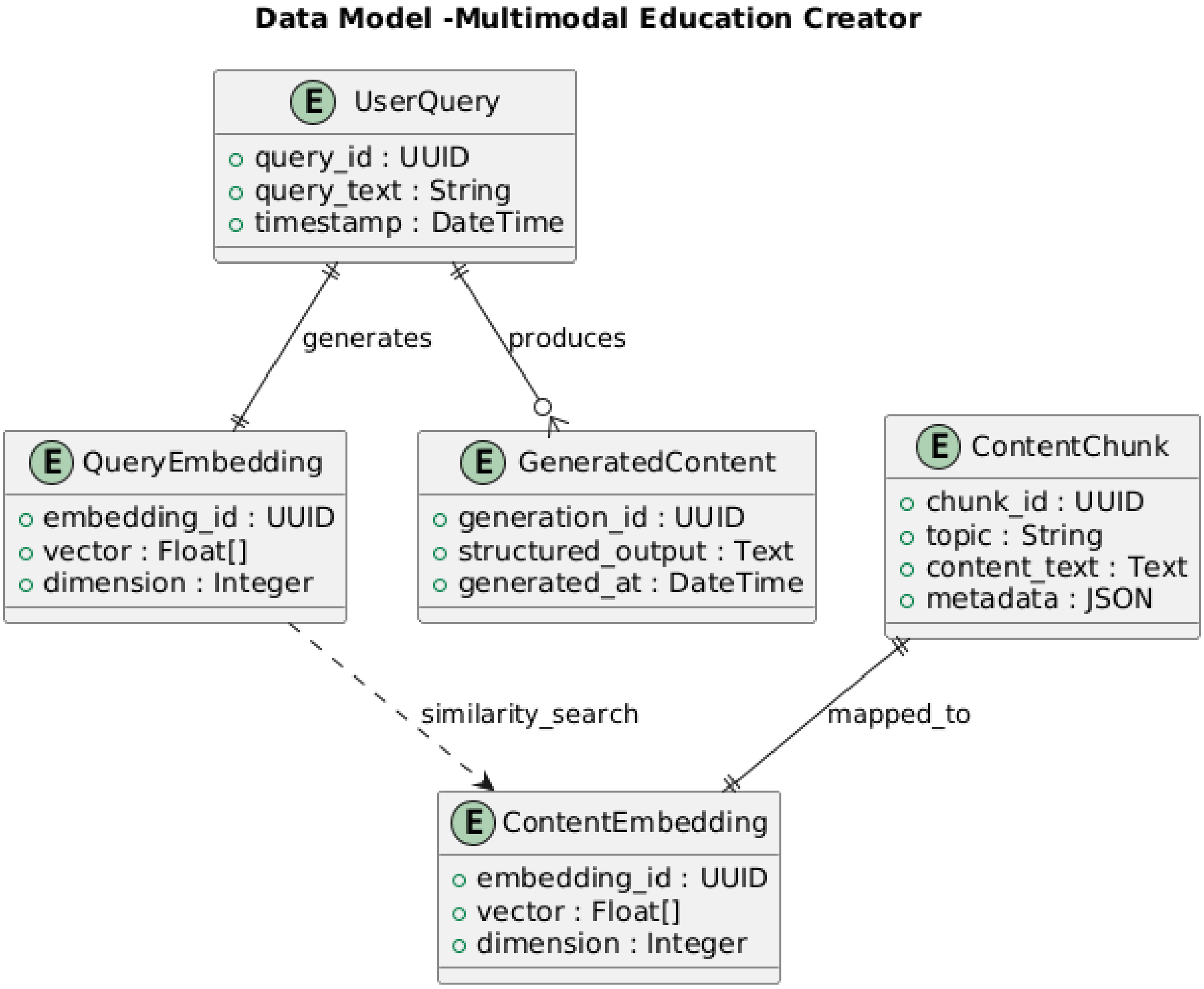
III. **Cache Status API:** Returns current cache usage details for performance monitoring**.**

- Endpoint: **GET /cache-status**

3. Data Design

3.1 Data Model

The data model is designed to support semantic retrieval and structured content generation. It separates user input, embeddings, stored knowledge, and generated output for modular and scalable processing.



Data Relationships:

- Each UserQuery generates one QueryEmbedding
- Each ContentChunk has one ContentEmbedding
- Query embeddings perform similarity search on content embeddings
- Retrieved content produces GeneratedContent

3.2 Data Access Mechanism

The system uses a vector-based data access mechanism to retrieve semantically relevant educational content. Instead of traditional SQL queries, the system performs embedding-based similarity search.

Data Storage Strategy

- Educational content is segmented into content chunks
- Each chunk has a precomputed embedding
- Stored in a serialized vector database (education_vector_db.pkl)
- Loaded into memory during application startup

Data Retrieval Process

- User query is converted into embedding
- Cosine similarity is computed against stored embeddings
- Top-K most relevant content chunks are retrieved
- Retrieved data is passed to content generation module

Access Characteristics

- Read-heavy design (retrieval-focused)
- In-memory vector comparison for low latency
- No direct modification of stored embeddings at runtime
- Stateless query-based access

Performance Optimization

- Precomputed embeddings reduce real-time computation
- Top-K filtering minimizes unnecessary data processing
- Optional caching for repeated queries

Security Considerations

- Controlled loading of serialized .pkl file
- Restricted file access permissions
- No permanent storage of user queries

3.3 Data Retention Policies

Data Lifecycle Process

Step 1 – **Data Collection**

- Educational materials are uploaded by administrator.
- User inputs are captured during active sessions.

Step 2 – **Processing**

- Content is chunked and converted into embeddings.
- Embeddings are stored in vector database.

Step 3 – **Active Usage**

- Data is accessed for similarity search and response generation.
- Query logs may be stored for analytics.

Step 4 – **Archival / Deletion**

- Old logs are automatically deleted after retention period.
- Dataset updates trigger embedding regeneration.
- Session-based content is cleared automatically.

Retention Rules:

- No sensitive personal data is permanently stored.
- User session data is cleared after logout or timeout.
- Logs older than 30–90 days are automatically purged.
- If dataset is replaced, old embeddings are deleted to prevent redundancy.
- Backup retention period: 7–14 days.

Security & Compliance Measures:

- Role-based access control for administrators.
- Encrypted storage for logs (if deployed in production).
- Regular backup and integrity checks.
- Compliance with general data protection principles

Data Deletion Policy

Data is deleted under the following conditions:

- Manual administrator request
- Dataset replacement
- System reset
- Expired retention period
- User account deletion

Deletion includes:

- Removing embeddings
- Clearing associated logs
- Cleaning temporary files

3.4 Data Migration

Data migration refers to the structured process of transferring educational datasets, embeddings, and system configurations from one environment to another

The system ensures safe migration of:

- Educational content datasets
- Vector embeddings
- Model configurations
- Application metadata

Migration Scenarios

- Initial dataset deployment
- Dataset updates (adding new study material)
- Migration between environments (local → cloud)
- Vector database upgrade or format change
- Backup restoration

Migration Process

Data Export

- Export existing dataset files
- Backup vector database (education_vector_db.pkl / FAISS index)
- Export configuration settings

Data Transfer

- Secure transfer using encrypted channels (SCP / SFTP / Cloud Storage)
- Maintain file integrity using checksum validation

Data Import

- Load dataset into new environment
- Regenerate embeddings if model version changes
- Validate vector database compatibility

Validation

- Run similarity search test queries
- Validate content retrieval accuracy
- Perform API health check

4. Interfaces

The system exposes structured interfaces to enable interaction between users, application modules, and external services. Interfaces are designed using REST principles and modular internal communication.

User Interface: Web-based interface (HTML / CSS / JS / Frontend Framework)

- Accept user topic input
- Display generated educational content
- Show error or system status messages
- Sends HTTP requests to backend APIs
- Receives JSON responses

Application Programming Interface (API): RESTful Web API, JSON request/response

- Accept user topic
- Trigger embedding generation
- Perform retrieval
- Return structured output

Core Endpoints:

- *POST /generate-content*
- *GET /health*
- *GET /cache-status* (optional)

Data Interface

Vector Database Interface:

- Loads. pkl or FAISS index file
- Performs cosine similarity search
- Returns Top-K relevant content chunks

File System Interface:

- Loads educational dataset
- Accesses stored embeddings
- Reads configuration files

External Service Interfaces

- Embedding Model API (e.g., local transformer or cloud model)
- Large Language Model API (for content generation)
- Optional Cloud Storage Service

Interface Characteristics

- Stateless REST communication
- JSON-based data exchange
- Secure HTTP (HTTPS) recommended
- Modular and loosely coupled design
- Easily extensible

5. State and Session Management

- **Stateless Backend Design:** Each user request (topic input) is processed independently without storing persistent session state on the server.

- **Session-Level Context Handling:** Temporary session memory can maintain user interaction history during active usage (if enabled).

- **In-Memory Runtime State:** Vector database and cache are loaded into memory during application runtime for faster processing.

- **Session Isolation:** User inputs are handled separately to prevent cross-session data leakage.

- **Future Enhancement:** Can integrate JWT-based authentication and server-side session storage for multi-user production deployment

6. Caching

Caching Strategy

- **Purpose of Caching:** Reduces repeated computation and improves response time for frequently requested topics.
- **Embedding Cache:** Stores previously generated query embeddings to avoid redundant vector computation**.**
- **Retrieval Result Cache:** Saves Top-K similarity search results for commonly searched educational topics**.**
- **In-Memory Caching Mechanism:** Fast access using in-memory storage for low-latency responses.
- **Scalable Extension:** Can be extended using distributed caching systems (e.g., Redis) in cloud deployment for high traffic handling**.**

Caching Mechanism in Educational Content Creator

- **Query Embedding Cache**: When a user enters a topic, its embedding is cached to avoid regenerating embeddings for repeated or similar queries.
- **Vector Retrieval Cache**: Frequently searched topics store their Top-K retrieved content chunks, reducing repeated similarity computations.
- **Generated Content Cache**: Final structured educational content for common topics is temporarily cached to improve response time.
- **In-Memory Storage**: Cache operates in-memory for low-latency access during runtime.
- **Future Scalability**: Can be extended using distributed caching (e.g., Redis) for cloud-based, high-traffic deployment.

7. Non-Functional Requirements

7.1 Security Aspects

Security is an essential consideration in the design of the AI-Based Educational Content Creator. Although the system operates primarily as a content generation and retrieval platform, it handles user input, backend processing, and vector-based knowledge storage. Therefore, appropriate security controls are implemented to ensure system integrity, data protection, and controlled access**.**

- **Input Validation and Sanitization:** Since the system accepts user-generated prompts through the web interface:

  - All input data is validated before processing.
  - Special characters and malicious patterns are sanitized.
  - Protection mechanisms are applied to prevent:
    - Injection attacks
    - Script-based manipulation
    - Unauthorized command execution

  This ensures the NLP pipeline processes only clean and structured input.

- **Secure Handling of Vector Database (.pkl):** The system uses a serialized vector database file (education_vector_db.pkl) to store embeddings and content metadata. Security measures include:

  - Restricted file access permissions Controlled loading of serialized object
  - Prevention of arbitrary file execution
  - Secure storage within the application directory

- **Application-Level Security:** The backend architecture ensures:

  - Separation of presentation layer and processing layer
  - Controlled interaction between web interface and NLP module
  - Encapsulation of embedding and retrieval logic

- **Authentication & Access Control (Future Enhancement):** For scalable or production deployment, the system can integrate:

  - User authentication (JWT-based session handling)
  - Role-based access control (RBAC)
  - API-level authorization tokens

- **Data Privacy Considerations:** This ensures minimal data retention risk.

  - No personally identifiable information (PII) is permanently stored.
  - User prompts are processed in-memory.
  - No external sharing of user data occurs.

- **Secure Deployment Practices:** For production-level deployment, recommended measures include**:**

  - HTTPS encryption for secure communication
  - Reverse proxy configuration (e.g., Nginx)
  - Containerization using Docker for isolation
  - Secure environment variable management
  - Firewall configuration on cloud servers

- **Model & Retrieval Security:** To prevent misuse of the AI pipeline:

  - Input size limits can be enforced
  - Rate limiting can be implemented
  - Logging and monitoring mechanisms can be added
  - Retrieval responses can be filtered for safe output formatting

- **Dependency and Environment Security:**

- Virtual environment (.venv) isolates dependencies
- requirements.txt ensures controlled package installation
- Regular updates prevent known vulnerabilities

7.2 Performance Aspects

- **Precomputed Embeddings:** Educational content embeddings are generated and stored in advance, reducing real-time computation overhead.
- **Efficient Similarity Search:** Semantic retrieval using cosine similarity enables fast Top-K relevant content fetching.
- **In-Memory Processing:** Vector database is loaded into memory for low-latency retrieval.
- **Modular Processing Pipeline:** Separation of NLP, retrieval, and generation improves execution efficiency.
- **Scalability Ready:** Can be optimized further using FAISS, caching mechanisms, parallel processing, and cloud deployment.

8. References

- Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* Advances in Neural Information Processing Systems (NeurIPS).
- Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* Proceedings of EMNLP-IJCNLP.
- Johnson, J., Douze, M., & Jégou, H. (2017). *FAISS: A library for efficient similarity search and clustering of dense vectors.* Facebook AI Research.
- FastAPI Documentation. https://fastapi.tiangolo.com/
- Hugging Face Transformers Documentation. https://huggingface.co/docs/transformers
- scikit-learn Documentation. https://scikit-learn.org/
- NumPy Documentation. https://numpy.org/
- Fielding, R. T. (2000).*Architectural Styles and the Design of Network-based Software Architectures (REST).* Doctoral Dissertation, University of California, Irvine.
- Designing Data-Intensive Applications, Kleppmann, M. (2017), O'Reilly Media.
- OpenAI API Documentation (for LLM integration concepts). https://platform.openai.com/docs