Afeka Academic College of Engineering

Department of Computer Science

Detailed Design Document

AI-Powered Chatbot for Academic Regulations

Project Team

Omri Roter 206502379

Niv Buskila 209507912

Amitay Manor 20863075

Project Supervisor

Dr. Sharon Yalov-Handzel

Fall Semester
2024

1. Introduction

The AI-Powered Academic Regulations Chatbot represents a sophisticated solution designed to revolutionize how Afeka College students access and understand academic regulations. This document outlines the comprehensive technical design that will enable the system to handle 1000 queries per day while maintaining 99% availability and ensuring response times under 5 seconds.

2. Architecture Design

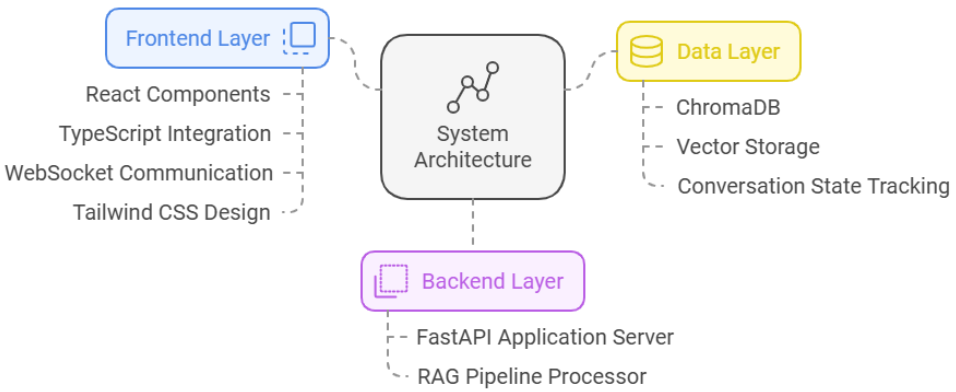
Our system implements an Event-Driven Architecture (EDA), carefully chosen to support real-time communication requirements and ensure system scalability. The architecture consists of three primary layers that operate independently while maintaining seamless communication through event streams.

The Frontend Layer - serves as the primary user interface, implementing *React* and *TypeScript* components that enable immediate user interaction. This layer utilizes *WebSocket* connections to maintain real-time communication channels with the backend services. Through *Tailwind CSS* implementation, we ensure a responsive design that adapts to various device formats while maintaining consistent user experience.

The Backend Layer - functions as the system's core processing unit, housing several critical components:

The FastAPI application server manages incoming events and coordinates responses across the system. This server implements advanced routing mechanisms to ensure optimal request distribution and processing.
The RAG pipeline processor, a crucial component of our backend architecture, handles the sophisticated task of processing user queries and generating contextually appropriate responses. This pipeline integrates seamlessly with our language model implementation to ensure accurate and relevant responses to student inquiries.

The Data Layer - utilizes *ChromaDB* as its foundation, implementing *vector storage* solutions that enable rapid information retrieval. This layer maintains sophisticated conversation state tracking mechanisms and handles data persistence across the system.



3. Data Storage Implementation

Our data storage strategy centers around *ChromaDB*, chosen for its superior vector storage capabilities and optimal performance characteristics. The system implements a multi-tiered storage approach that ensures both data integrity and rapid access to frequently requested information.

The vector storage system maintains embeddings of regulatory documents, enabling semantic search capabilities that significantly improve response accuracy. Document versioning mechanisms track changes to regulations over time, ensuring students always receive the most current information while maintaining access to historical versions when necessary.

The conversation state management system tracks ongoing interactions, enabling context-aware responses that enhance the natural flow of communication. This system maintains temporary conversation states in memory while persistently storing critical interaction data for future reference and system improvement.

4. Data Description

Our system utilizes a sophisticated data organization structure to manage academic regulations, embeddings, and search contexts efficiently. Below we detail the core data structures and their relationships.

Academic Regulations Table This primary table stores all regulatory documents and their associated metadata:

- RegulationID: Unique identifier for each regulation (UUID)
- Title: Descriptive title of the regulation (VARCHAR(255))
- Content: Full text content of the regulation (TEXT)
- Version: Version number of the regulation (INTEGER)
- LastUpdateTimestamp: Time of last modification (TIMESTAMP)
- Category: Classification of regulation type (VARCHAR(100))
- Department: Associated academic department (VARCHAR(100))
- Status: Current status of the regulation (ACTIVE/ARCHIVED)
- ApprovalDate: Date of regulation approval (DATE)

Vector Embeddings Table Stores the processed embeddings used for semantic search:

- ChunkID: Unique identifier for text chunk (UUID)
- DocumentReference: Reference to source regulation (FOREIGN KEY)
- TextContent: Original text segment (TEXT)
- Embedding: 1536-dimensional vector representation (VECTOR)
- Position: Position within source document (INTEGER)
- SectionInfo: Structural location information (JSON)
- CreationDate: Date of embedding creation (TIMESTAMP)

Search Context Table Maintains information about search operations and performance:

- QueryID: Unique identifier for each query (UUID)
- OriginalQuery: User's input text (TEXT)
- QueryEmbedding: Vector representation of query (VECTOR)
- RelevantChunks: References to matched chunks (ARRAY)
- SimilarityScores: Matching scores for results (ARRAY)
- ResponseTime: Query processing duration (INTEGER)
- Timestamp: Time of query (TIMESTAMP)

5. API Specification

Our API architecture implements RESTful principles while supporting WebSocket connections for real-time interactions. Key endpoints include:

Query Processing Endpoints

- POST /api/v1/query
    - Handles new user queries
    - Accepts: Query text, conversation context
    - Returns: Formatted response, source references
- POST /api/v1/feedback
    - Processes user feedback
    - Accepts: Query ID, feedback data
    - Returns: Acknowledgment status

Document Management Endpoints

- POST /api/v1/documents
    - Uploads new regulations
    - Accepts: Document content, metadata
    - Returns: Document ID, processing status
- PUT /api/v1/documents/{id}
    - Updates existing regulations
    - Accepts: Updated content, version data
    - Returns: Updated document status

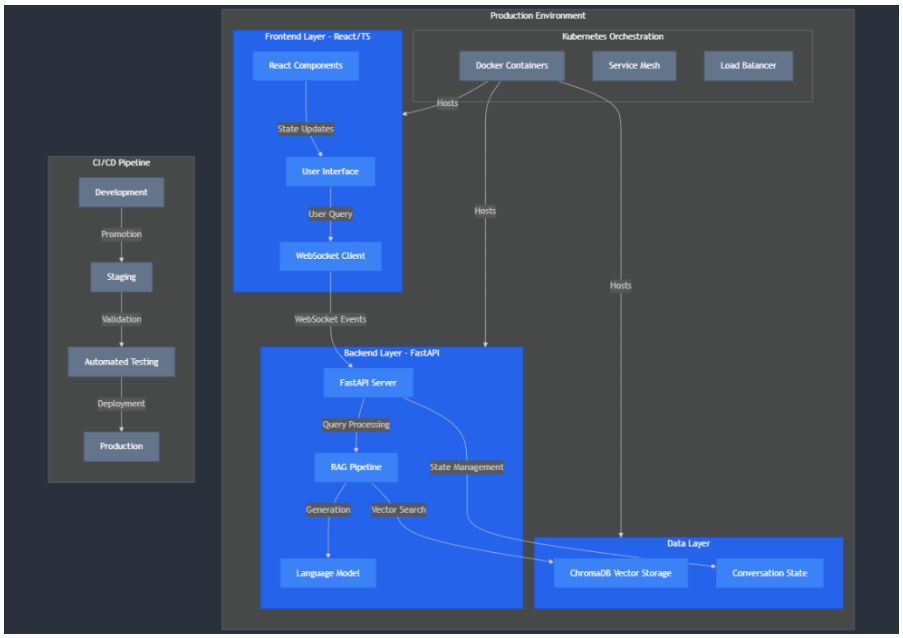
6. Interface Design

The interface implementation utilizes modern web technologies to ensure optimal performance and user experience:

Development Environment Our technology stack comprises:

- Frontend: React 18.0+ with TypeScript 4.9+
- Backend: Python 3.10+ with FastAPI
- Database: ChromaDB for vector storage
- Infrastructure: Docker containers orchestrated by Kubernetes

The development environment supports continuous integration and deployment through automated testing and build processes. We maintain separate development, staging, and production environments to ensure code quality and system stability.



7. Algorithm Description

The system's core algorithmic components focus on efficient query processing and accurate response generation:

RAG Pipeline Implementation the Retrieval Augmented Generation pipeline consists of three primary stages:

1. Query Processing
    - Input normalization and cleaning
    - Context extraction and enhancement
    - Query vector generation using embedding model
    - Time Complexity: O(n) where n is query length
2. Document Retrieval
    - Vector similarity search in ChromaDB
    - Context window optimization
    - Relevant chunk selection and ranking
    - Time Complexity: O(log n) for indexed searches
3. Response Generation
    - Context assembly and formatting
    - Language model prompt construction
    - Response validation and filtering
    - Time Complexity: O(n) where n is response length

Performance Optimization The system implements several optimization strategies:

- Parallel processing of vector searches
- Batch processing of document updates
- Response streaming for long-form content