EduSync AI Classroom Management WebApp

An AI-powered platform for modern classroom management and educational synchronization

Steps to Run The Application

- **Clone the repository**

```
1) git clone https://github.com/mirajdeepbhandari/EduSycn_AI_Classroom_Management_WebAPP.git
```

- **Create virtual environment**

```
2) python -m venv ./myvenv
```

- ⚡ **Activate the environment**

```
3) In Windows CMD:
myvenv\Scripts\activate.bat
```

- **Install dependencies**

```
4) pip install -r requirements.txt
```

- **Configure API key**

```
5) Create a .env file in the root directory and add:
  GOOGLE_API_KEY=your_api_key_here
```

Get your API key from Google AI Studio (https://aistudio.google.com/apikey)

- **Setup sentiment analysis model**

```
6) clone the model repo go inside the model main folder move the inner sentimentModel directory to the static folder of project
   git clone https://huggingface.co/mirajbhandari/sentimentModel
```

- **Install database server**

Download and install XAMPP (https://www.apachefriends.org/download.html)

- **Configure database**

```
7)  Create database named 'edusync_f'
8) Import the edusync_f.sql file
```

- **Start services**

```
9) Start XAMPP Apache and MySQL services
```

- **Run the server**

```
10) uvicorn main:app
```

About the Directory Structure

The application follows the MVC pattern with custom components and a modified directory structure to integrate web components, AI components, and database into a single application.

- **authentication**

  Contains user authentication controls including login status verification (auth_required) and role-based access controls (teacher, student, admin). Uses FastAPI sessions to authenticate user ID and roles, redirecting unauthorized users to login or error pages.

- **Docs_Useful_items**

  Contains flowchart diagrams, LangGraph agentic architectures, and other necessary documentation including server run code and system details.

- **FypVenv**

  Virtual environment containing all necessary modules and packages for the application.

- **graphs**

  Controls content generation workflow through LangGraph stateful structuration. The graph.py module describes an agentic system that orchestrates multiple specialist tools (introduction writer, body content generator, conclusion writer, PPT compiler) in an ordered state graph.

- **jsonDB**

  JSON database storage containing organized education data files like mcq_data.json where lesson content is stored in nested JSON format. Used for storing and displaying student MCQ exams with automatic marking.

- **models**

  Application data layer containing ORM definitions and database schemas. Includes pre-defined structures for system objects (assignments, users, classrooms) that map to database tables, using SQLAlchemy for type safety and relationship management.

- **process**

  Manages data processing and content transformation, primarily extracting, structuring, and refining data from PDF documents for use in the classroom management system. Supports asynchronous and batch processing.

- **routes**

  API endpoint nexus containing route files for different functionalities including authentication, classroom actions, assignment management, real-time features, and role-based dashboards, following RESTful principles.

- **schemas**

  Defines data structures and validation rules for the application, particularly for LLM content generation and processing. Uses Pydantic's BaseModel to define strict templates for different elements of slides.

- **services**

  Contains AI core functionality and real-time communications:

  - **AIServices**: Intelligent features like MCQ generation, slide generation, and content summarization
  - **WebSockets**: Live conversation and notification functionality

- **state**

  Stores conversations between different nodes in the agentic framework, essential for LangGraph in slide generation.

- **static**

  Contains application static resources (images, CSS, JavaScript, fonts) that are sent directly to clients.

- **templates**

  Contains HTML templates for generating dynamic web pages, typically used with a templating engine like Jinja2.

- **tools**

  Contains tools required by agents in Langraph for slide generation, using schemas to sync with agents and generate different presentation sections.

- **utils**

  Support modules and utility functions for operations across the application, including data formatting, validation, and preprocessing.

- **main.py**

Application entry point that configures and starts the FastAPI server. Initializes middlewares like sessions and CORS, mounts static and template files, and imports route modules. Defines base API configuration and security policies.