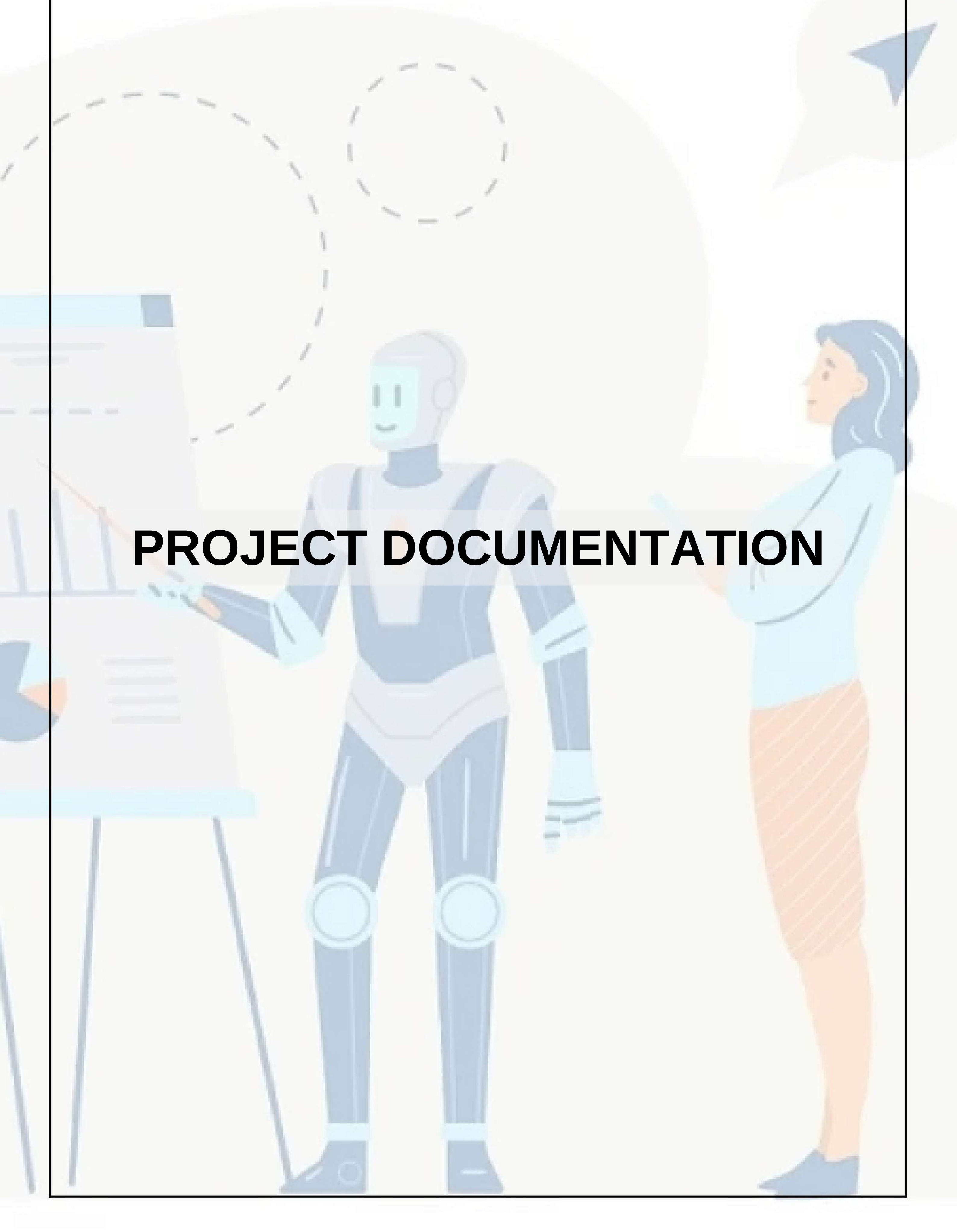


PROJECT DOCUMENTATION

Introduction

EduTutor AI: Personalized Learning with Generative AI and LMS Integration

1. Project Overview

Purpose:

EduTutor AI is a smart education platform that puts the needs of each student at the heart of learning. Using the power of artificial intelligence, it creates personalized quizzes based on how a student is performing, ensuring that no two learning paths are exactly the same. Students receive real-time feedback on their answers, helping them quickly understand their strengths and the areas where they need more practice. It’s like having a personal tutor that’s always available and adapts as they grow.

But EduTutor AI isn’t just for students—it’s a powerful tool for teachers too. With seamless Google Classroom integration, educators can easily assign work, track progress, and manage everything from a single place. A user-friendly dashboard brings all important data together, offering insights into each student’s progress over time. Whether you're learning or teaching, EduTutor AI makes the experience smoother, more engaging, and truly personalized.

The overall goal of the project is to enhance educational outcomes by:

- Delivering AI-generated, adaptive quizzes using IBM Watsonx Granite models.
- Providing insightful analytics for educators to monitor and personalize instruction.
- Ensuring real-time interaction through a modular system architecture involving FastAPI, Pinecone Vector DB, and Streamlit frontend.

Features:

1 . AI-Powered Personalized Quizzes :

- Dynamically generates quizzes tailored to each student's performance using IBM Watsonx Granite models.

2 . Real-Time Feedback & Adaptive Learning :

- Instantly evaluates student answers and adjusts quiz difficulty to match learning progress.

3 . Educator Dashboard & Performance Analytics :

- Provides teachers with real-time insights into student performance, quiz history, and topic-wise progress.

4 . Google Classroom Integration :

- Seamlessly syncs courses and student data, enabling automatic quiz generation aligned with classroom content.

5 . Modular, Scalable Architecture :

- Built using FastAPI backend, Streamlit frontend, Pinecone vector DB, and LangChain AI integration for efficient, scalable performance.

2. Architecture

Frontend

- **Technology:** Streamlit
- **Role:** The frontend of EduTutor AI is built using Streamlit, a lightweight Python-based web framework ideal for data-driven interfaces. & Provides a user-friendly interface for both students and educators.
- **Features:**
  1. **Role-based dashboards (Student / Educator) :**
     ~ Student Panel: Login (manual or Google), quiz dashboard, take quiz, view quiz history.
     ~ Educator Panel: View student analytics, quiz history, performance summaries.
  2. **Quiz submission forms :**
     ~ Quizzes are dynamically loaded from backend APIs.
     ~ Submissions trigger instant evaluation and feedback display.
  3. **Login system (Manual & Google OAuth)** :
     ~ Uses google-auth-oauthlib to enable secure Google login.

Backend

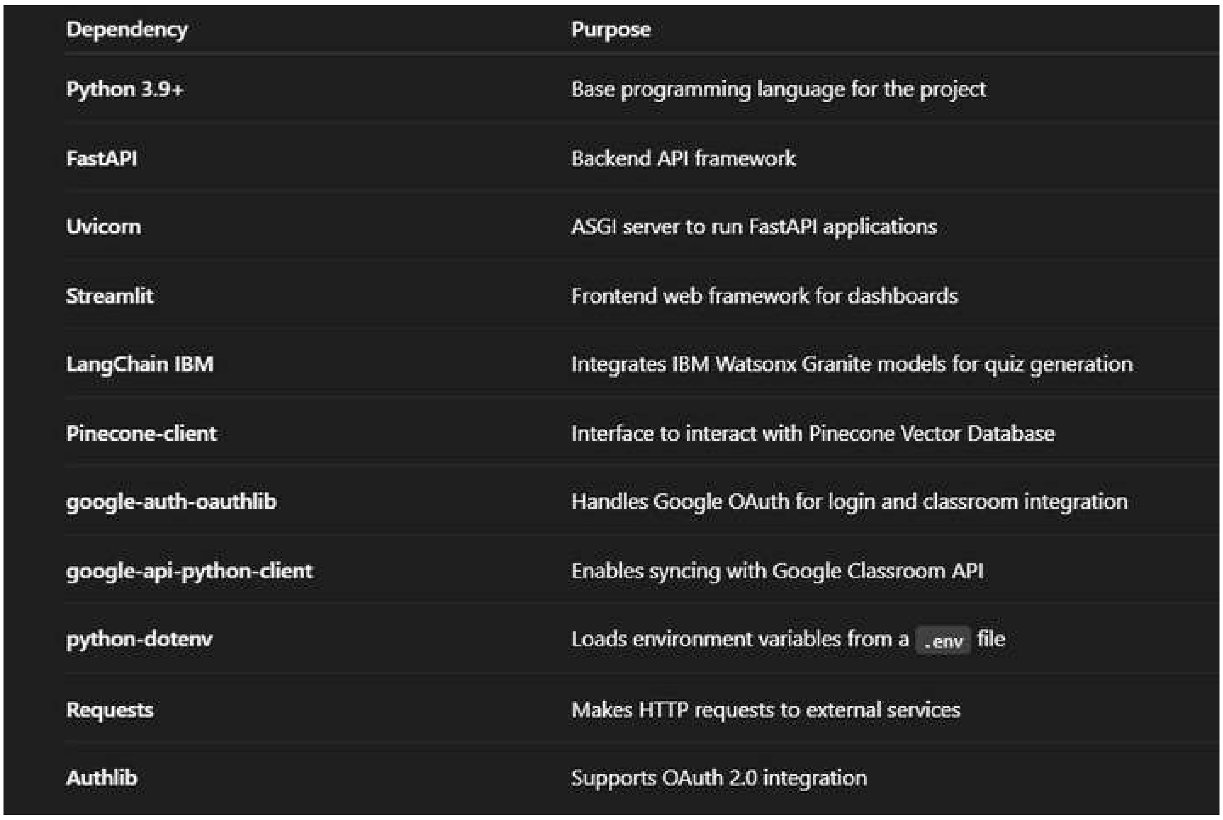
- **Technology:** FastAPI
- **Role:** Handles all core logic and API services.
- **Features:**
  1. **User authentication and role management :**
     ~ Handles user login, session management, and role-based access.
  2. **Quiz generation and answer evaluation using IBM Watsonx:**
     ~ Integrates with IBM Watsonx Granite LLM via LangChain to dynamically generate quizzes based on topics.
  3. **Integration with Google Classroom and metadata updates**:
     ~ Uses google-api-python-client to fetch student courses and subjects.
     ~ Updates quiz attempt data, scores, and progress into the Pinecone vector DB.

Database & Storage

- **Technology :** Pinecone Vector Database
- **Role:** EduTutor AI uses Pinecone, a vector database designed for similarity search and fast retrieval of high-dimensional embeddings. Stores and manages user embeddings, quiz metadata, and performance insights.
- **Features:**
  1. Similarity search for adaptive learning
  2. Storing quiz history, scores, topics, and timestamps
  3. Supporting personalized recommendations and feedback

3. SETUP INSTRUCTIONS

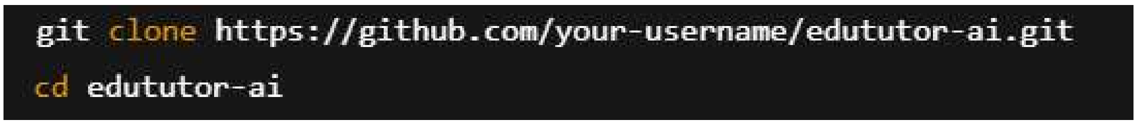
Prerequisites:



Dependency	Purpose
Python 3.9+	Base programming language for the project
FastAPI	Backend API framework
Uvicorn	ASGI server to run FastAPI applications
Streamlit	Frontend web framework for dashboards
LangChain IBM	Integrates IBM Watsonx Granite models for quiz generation
Pinecone-client	Interface to interact with Pinecone Vector Database
google-auth-oauthlib	Handles Google OAuth for login and classroom integration
google-api-python-client	Enables syncing with Google Classroom API
python-dotenv	Loads environment variables from a `.env` file
Requests	Makes HTTP requests to external services
Authlib	Supports OAuth 2.0 integration

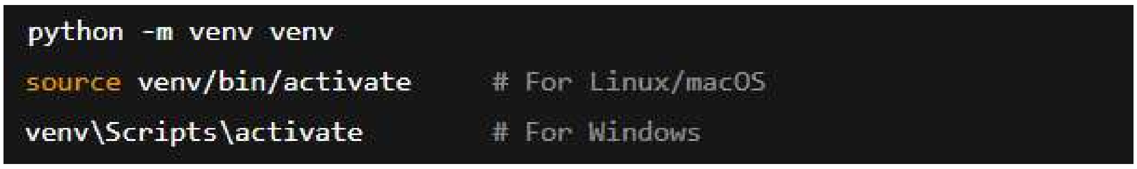
4. Installation:

Step 1: Clone the Repository



```
git clone https://github.com/your-username/edututor-ai.git
cd edututor-ai
```

Step 2: Create a Virtual Environment



```
python -m venv venv
source venv/bin/activate      # For Linux/macOS
venv\Scripts\activate         # For Windows
```

Step 3: Install Dependencies

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

This installs all necessary packages including:

- FastAPI
- Streamlit
- LangChain
- Pinecone-client
- Google API libraries
- IBM Watsonx integrations

STEP 4: SET UP ENVIRONMENT VARIABLES

- **Create a .env file in the project root and add the following variables:**

```
WATSONX_MODEL_ID=granite-13b-instruct-v2
WATSONX_API_KEY=your_ibm_watsonx_api_key
WATSONX_ENDPOINT=https://us-south.ml.cloud.ibm.com
WATSONX_PROJECT_ID=your_ibm_project_id
PINECONE_API_KEY=your_pinecone_api_key
PINECONE_INDEX_NAME=edututor
```

STEP 5: Final Step: Run the Application

- **Run the Backend (FastAPI with Uvicorn)**

```
uvicorn backend.main:app --reload
```

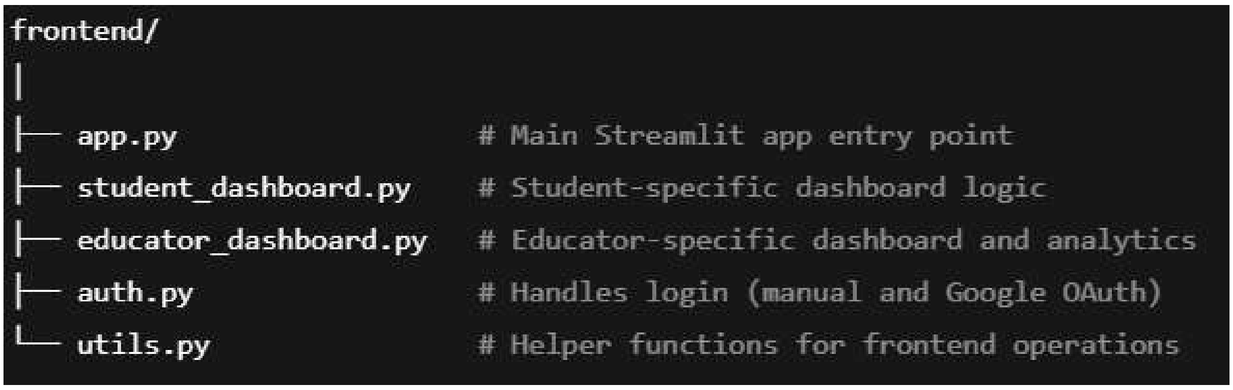
- **Run the Frontend (Streamlit)**

```
streamlit run frontend/app.py
```

5. Folder Structure

- **Client – Streamlit Frontend**

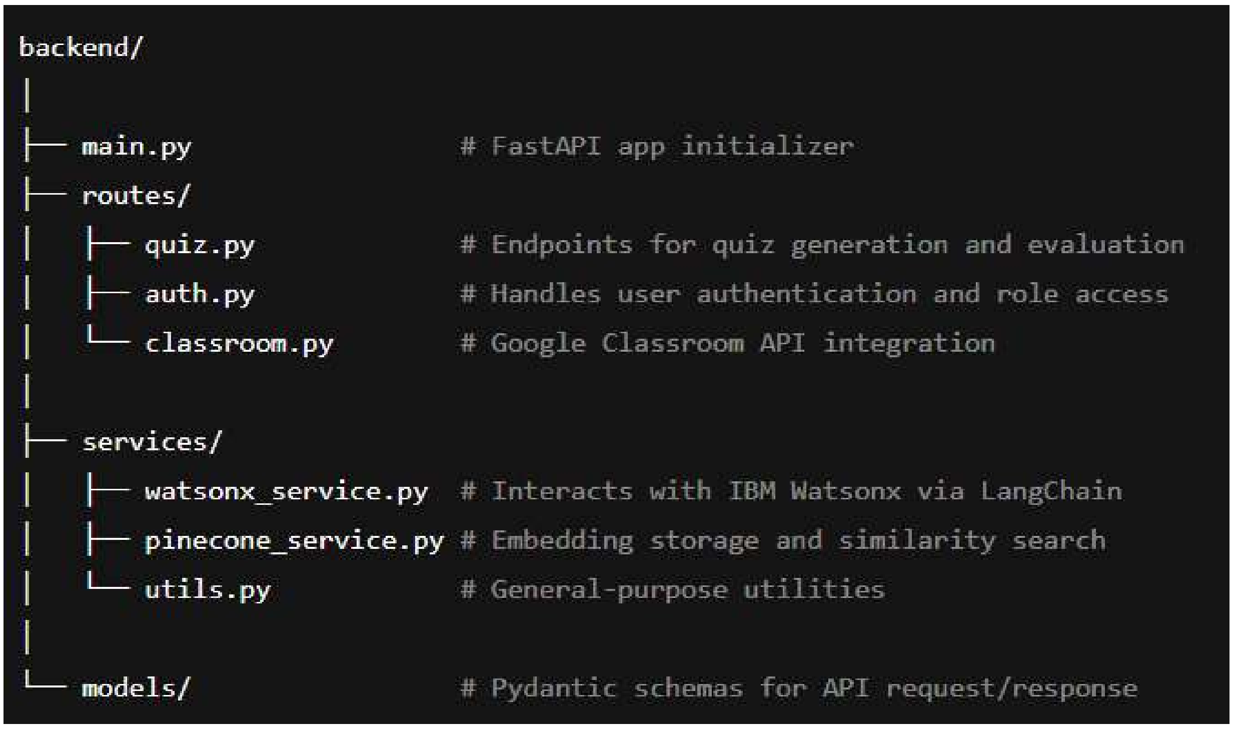
Folder: /frontend/
Purpose: Contains all the code related to the user interface and role-based dashboards.
Structure:



```
frontend/
│
├── app.py                  # Main Streamlit app entry point
├── student_dashboard.py    # Student-specific dashboard logic
├── educator_dashboard.py   # Educator-specific dashboard and analytics
├── auth.py                 # Handles login (manual and Google OAuth)
└── utils.py                # Helper functions for frontend operations
```

- **Server – FastAPI Backend**

Folder: /backend/
Purpose: Contains API endpoints, AI model integration, Google Classroom sync, and Pinecone DB handling.
Structure:



```
backend/
│
├── main.py                # FastAPI app initializer
├── routes/
│   ├── quiz.py            # Endpoints for quiz generation and evaluation
│   ├── auth.py            # Handles user authentication and role access
│   └── classroom.py       # Google Classroom API integration
│
├── services/
│   ├── watsonx_service.py # Interacts with IBM Watsonx via LangChain
│   ├── pinecone_service.py # Embedding storage and similarity search
│   └── utils.py           # General-purpose utilities
│
└── models/                # Pydantic schemas for API request/response
```

6. Running the Application

- **Run the Backend (FastAPI with Uvicorn)**

```
uvicorn backend.main:app --reload
```

- **Run the Frontend (Streamlit)**

```
streamlit run frontend/app.py
```

7. API Documentation

The FastAPI backend exposes the following RESTful API endpoints for authentication, quiz generation, evaluation, and classroom integration.

1. User Authentication

POST /auth/login

- **Description:** Authenticate a user manually.
- **Request Body (JSON):**

```
{
  "username": "navya",
  "password": "your_password"
}
```

2. Google OAuth Login

GET /auth/google-login

- **Description:** Initiates Google OAuth flow.
- **Response:**

~ Redirects to Google OAuth consent screen.

GET /auth/google-callback

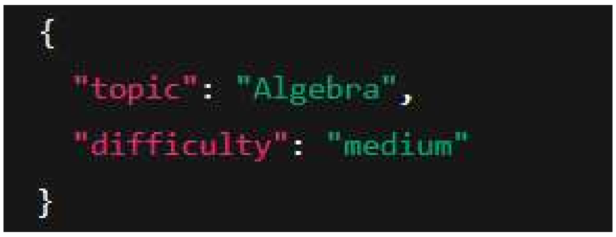
- **Description:** Handles Google login callback and token exchange.
- **Response:**

```
{
  "access_token": "ya29.a0AVvZ...",
  "user_email": "user@gmail.com",
  "role": "educator"
}
```

3. Quiz Generation

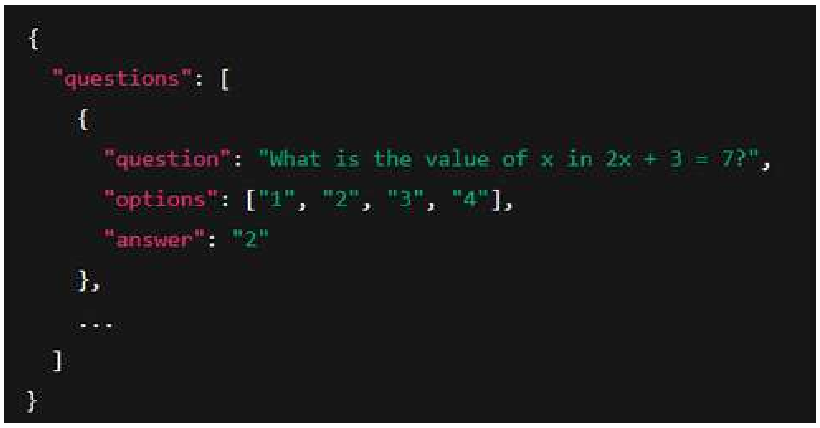
POST /quiz/generate

- **Description:** Generates a quiz based on the provided topic using IBM Watsonx**.**
- **Request Body (JSON):**



```
{
  "topic": "Algebra",
  "difficulty": "medium"
}
```

- **Response:**

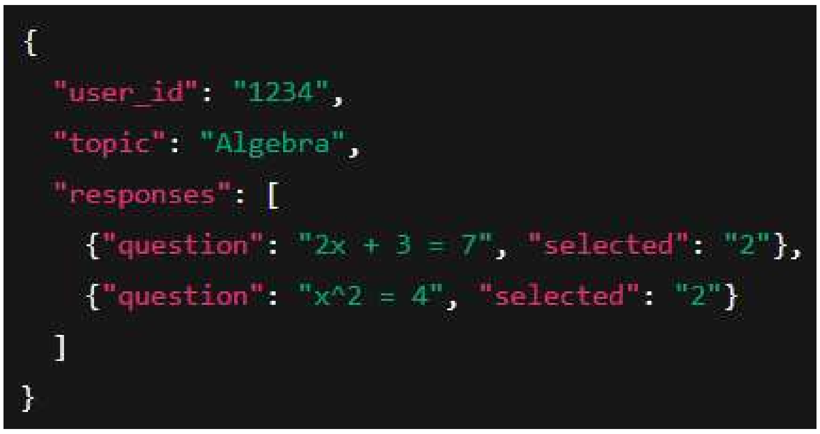


```
{
  "questions": [
    {
      "question": "What is the value of x in 2x + 3 = 7?",
      "options": ["1", "2", "3", "4"],
      "answer": "2"
    },
    ...
  ]
}
```

4. Quiz Submission and Evaluation

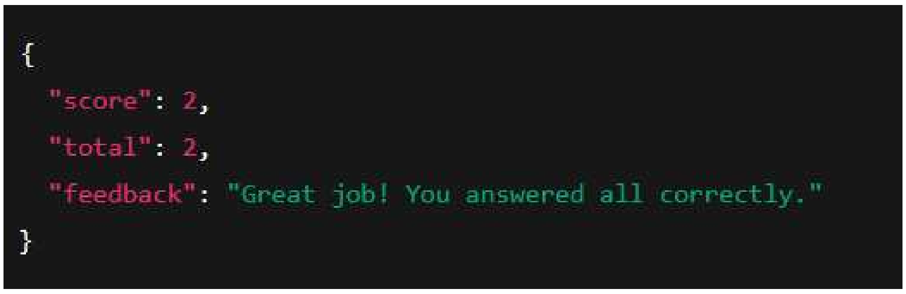
POST /quiz/submit

- **Description:** Submits quiz answers and returns evaluation.
- **Request Body (JSON):**



```
{
  "user_id": "1234",
  "topic": "Algebra",
  "responses": [
    {"question": "2x + 3 = 7", "selected": "2"},
    {"question": "x^2 = 4", "selected": "2"}
  ]
}
```

- **Response:**

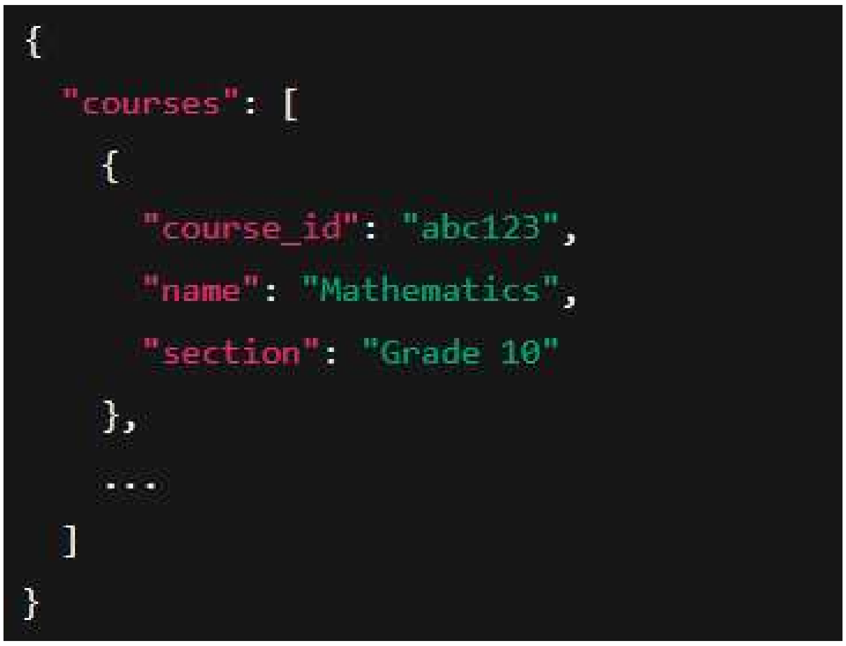


```
{
  "score": 2,
  "total": 2,
  "feedback": "Great job! You answered all correctly."
}
```

5. Google Classroom Sync

GET /classroom/sync

- **Description:** Syncs classroom data using authenticated Google account.
- **Headers:**
  ~ Authorization: Bearer <access_token>
- **Response:**

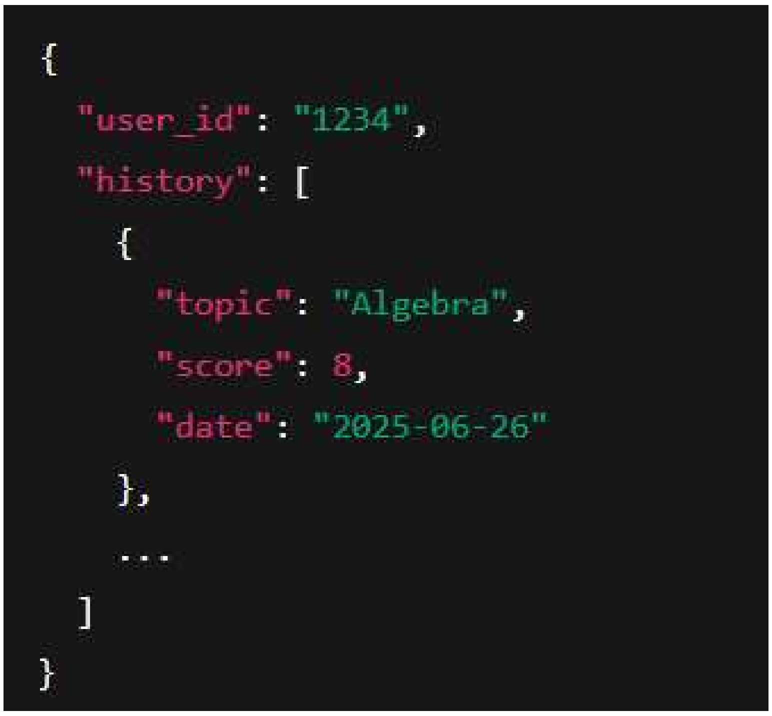


```
{
  "courses": [
    {
      "course_id": "abc123",
      "name": "Mathematics",
      "section": "Grade 10"
    },
    ...
  ]
}
```

6. Fetch Student Performance

GET /student/performance/{user_id}

- **Description:** Fetch performance history for a student.
- **Path Parameter:**
  ~ user_id : ID of the student.
- **Response:**



```
{
  "user_id": "1234",
  "history": [
    {
      "topic": "Algebra",
      "score": 8,
      "date": "2025-06-26"
    },
    ...
  ]
}
```

8. Authentication

EduTutor AI uses secure authentication and role-based authorization to ensure controlled access for students and educators.

- **Authentication Methods User**

  1 . Manual Login (Username & Password)

  ~ Users can log in with a registered username and password.
  ~ Upon successful login, a JWT (JSON Web Token) is issued to the user.

  2 . Google OAuth 2.0 Login

  ~ Powered by google-auth-oauthlib, users can log in using their Google accounts.
  ~ Once authenticated via Google, a secure session is created, and user details (email, role) are fetched.

- **Token-Based Authentication (JWT)**

  ~ After successful login (manual or OAuth), a JWT token is issued.

  This token:

  ~ Is included in the Authorization header of every API request.
  ~ Encodes the user's identity and role (e.g., student, educator).
  ~ Has an expiration time to enhance security.

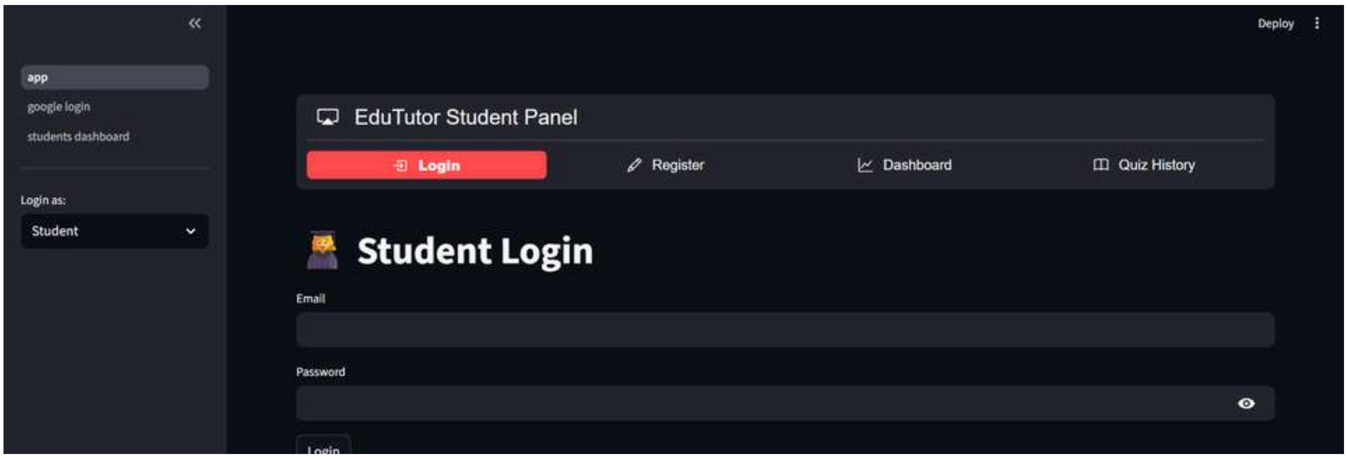
- **Authorization & Role Management**

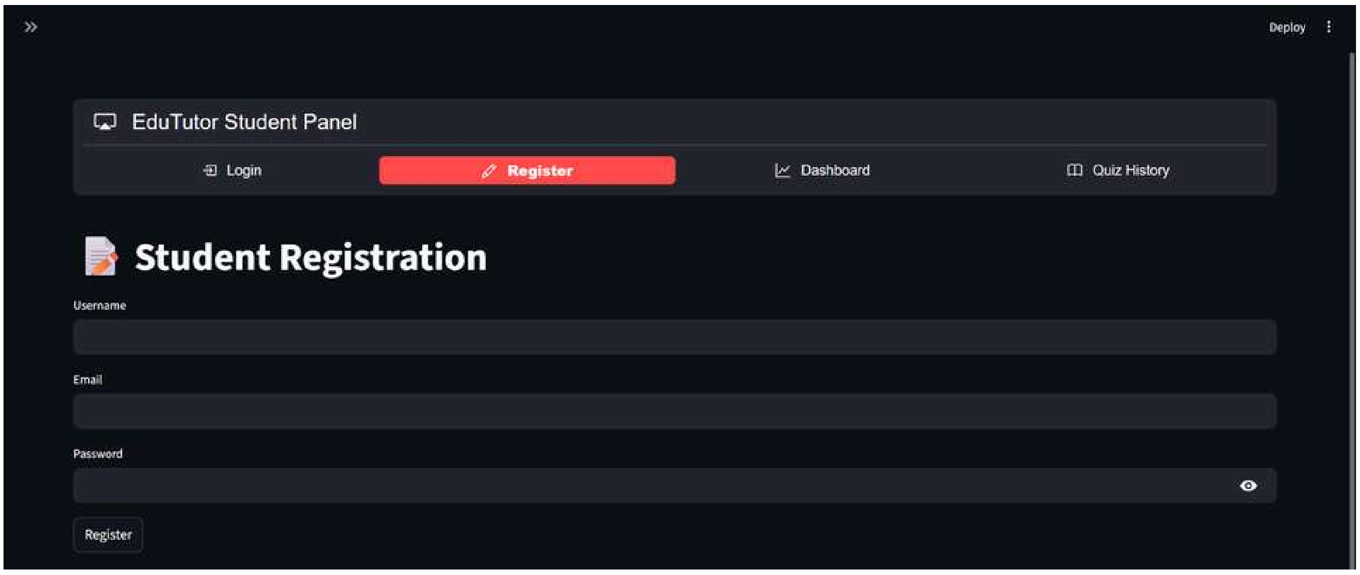
  Each user is assigned a role:
  ~ **student** – can access quiz dashboards, take quizzes, view history.
  ~ **educator –** can view all student data, performance analytics, and classroom sync. Role-based access is enforced using dependency injection in FastAPI routes.

- **Session Handling**

  ~ JWTs are stateless, so the backend doesn't store session information.
  ~ Users are considered "logged in" as long as they hold a valid, non-expired token.
  ~ For Google OAuth, session info is cached during the redirect and token exchange phases.

9. User Interface





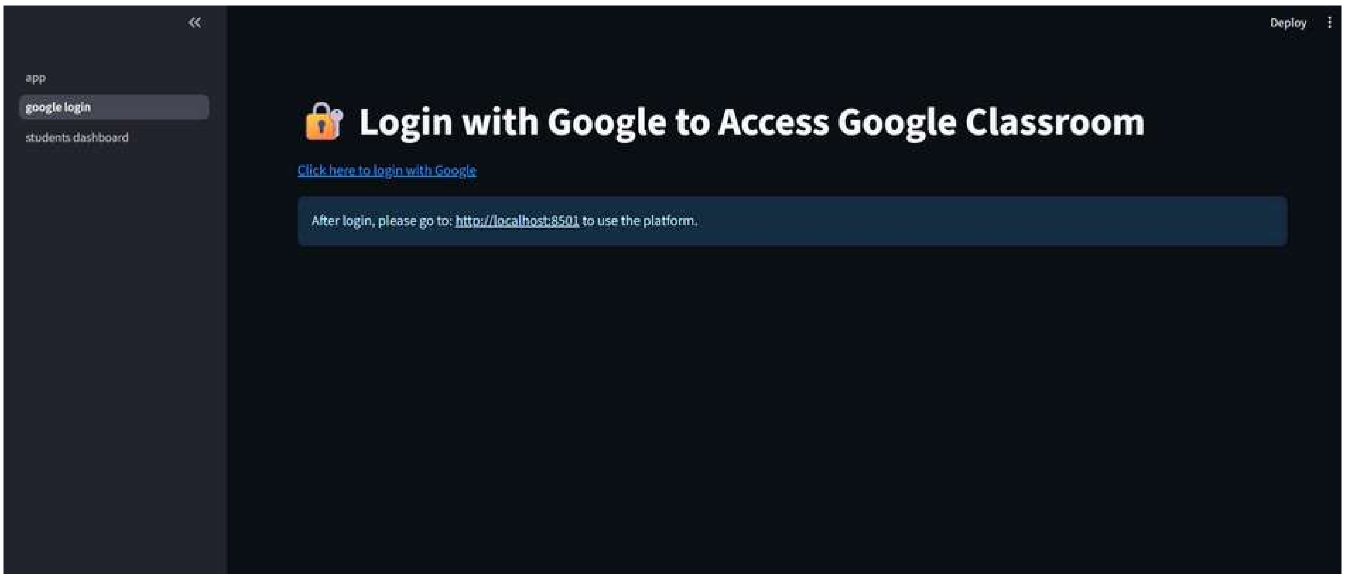
Student Registration

Username

Email

Password

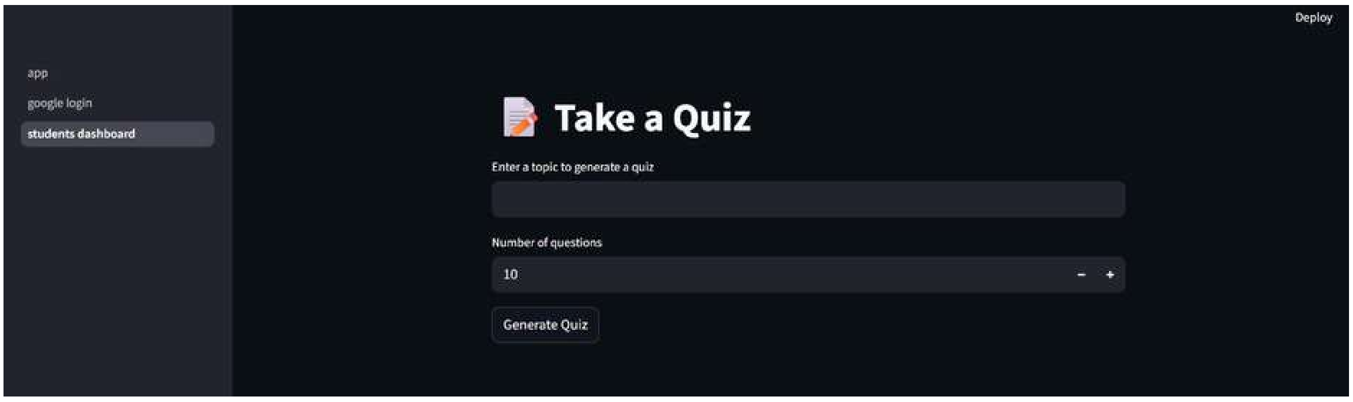
Register



Login with Google to Access Google Classroom

Click here to login with Google

After login, please go to: http://localhost:8501 to use the platform.



Take a Quiz

Enter a topic to generate a quiz

Number of questions

10

Generate Quiz

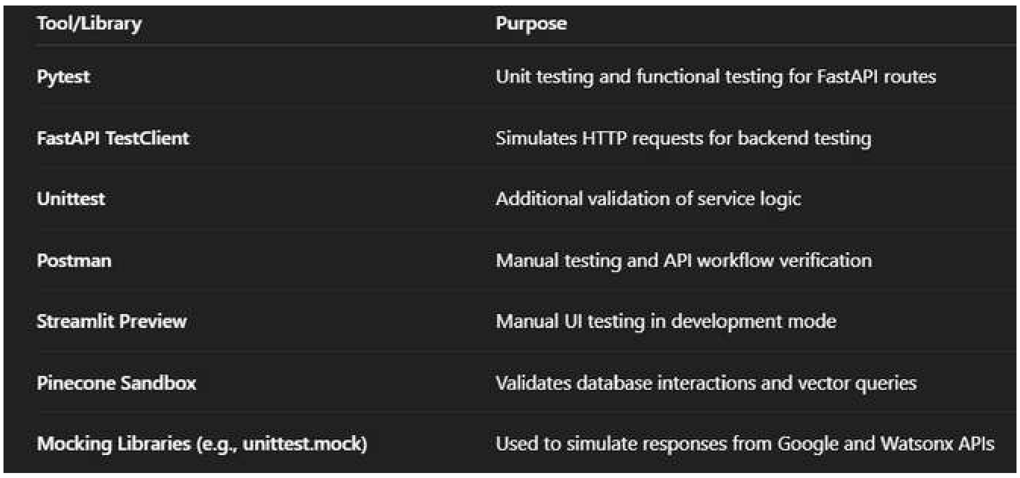
10. Testing

To ensure reliability, performance, and correctness of the EduTutor AI platform, a comprehensive testing strategy was adopted across backend APIs, frontend components, and integration points.

Testing Goals

- Validate core functionalities (quiz generation, evaluation, login)
- Ensure role-based access control is enforced correctly
- Confirm integration with external services (IBM Watsonx, Google Classroom, Pinecone)
- Detect edge cases and handle invalid inputs gracefully
- Maintain code quality and prevent regression through automated testing

Tools Used :



Tool/Library	Purpose
Pytest	Unit testing and functional testing for FastAPI routes
FastAPI TestClient	Simulates HTTP requests for backend testing
Unittest	Additional validation of service logic
Postman	Manual testing and API workflow verification
Streamlit Preview	Manual UI testing in development mode
Pinecone Sandbox	Validates database interactions and vector queries
Mocking Libraries (e.g., unittest.mock)	Used to simulate responses from Google and Watsonx APIs

Types of Testing Performed

1. **Unit Testing**
   ~ Focused on isolated functions (e.g., quiz parsing, token decoding).
2. **API Testing**
   ~ Used TestClient from FastAPI and Postman to test:
   - Authentication endpoints
   - Quiz generation
   - Quiz submission and scoring
   - Classroom syncing
3. **Integration Testing**
   ~ Verified how the backend communicates with:
   - IBM Watsonx (LLM model outputs)
   - Pinecone (vector similarity and metadata handling)
   - Google Classroom (OAuth and course sync)
4. **Frontend Manual Testing**
   ~ Ensured Streamlit UI behaves correctly across different user roles.
   ~ Verified Google login flow and dashboard updates.