

Project Report:

EduTutor AI : Personalized Learning with Generative AI and LMS Integration

1. INTRODUCTION

1.1 Project Overview

EduTutor AI is an AI-powered personalized education platform that revolutionizes student learning and educator assessment. It dynamically generates quizzes, evaluates students, integrates with Google Classroom, and provides real-time feedback using Google Flan T5 large model. Its modular architecture ensures seamless scalability and efficient personalized learning delivery.

1.2 Purpose

The purpose of EduTutor AI is to bridge learning gaps, enhance student engagement, and assist educators in tracking performance insights efficiently by leveraging generative AI and integrated LMS functionalities.

2. IDEATION PHASE

2.1 Problem Statement

Traditional online learning platforms lack personalized assessment and adaptive feedback, limiting effective learning outcomes. EduTutor AI solves this by dynamically generating quizzes tailored to each student's performance and academic needs.

2.2 Empathy Map Canvas

SAYS - quotes or statements, keywords or phrases they commonly use

THINKS - what occupies their thoughts, what are their worries and aspirations?

DOES - behaviour patterns, daily routines or activities

FEELS - how do they feel about their experiences, what worries or excites them?

PAINS - obstacles faced, risks or negative experiences

GAINS - goals, needs, or measures of success, benefits they seek

2.3 Brainstorming

- AI for quiz generation using Google-Flan-T5-large
- Pinecone DB for storing embeddings of academic topics
- Google Classroom API for data sync
- React for frontend for quick deployment and interactivity

3. REQUIREMENT ANALYSIS

3.1 Customer Journey Map

1. **Login** with Google Classroom credentials.
2. **Course Sync** – fetch subjects and class details.
3. **Diagnostic Test** – adaptive quizzes generation.
4. **Performance Evaluation** – real-time feedback and recommendations.
5. **Educator Dashboard** – teachers access insights to personalize teaching.

3.2 Solution Requirement

- **Functional:** Google OAuth login, AI quiz generation, performance evaluation, educator dashboard, Pinecone integration
- **Non-functional:** Fast response (< 2s for quiz generation), secure OAuth flow, scalable backend APIs

3.3 Technology Stack

Component	Technology
Backend AI	Google-flan-t5-large
Database	Pinecone Vector DB
Frontend	React
Integration	Google Classroom API
Programming	Python

4. PROJECT DESIGN

4.1 Problem Solution Fit

Students need adaptive quizzes and personalized feedback to improve learning efficiency. EduTutor AI delivers this using IBM generative AI models integrated with LMS data.

4.2 Proposed Solution

- Use Google Classroom API for course data
- Generate quizzes dynamically with google flan t5 large
- Evaluate responses for instant feedback
- Pinecone DB stores vector embeddings for efficient topic search
- React frontend for user interaction

5. PROJECT PLANNING & SCHEDULING

5.1 Project Planning

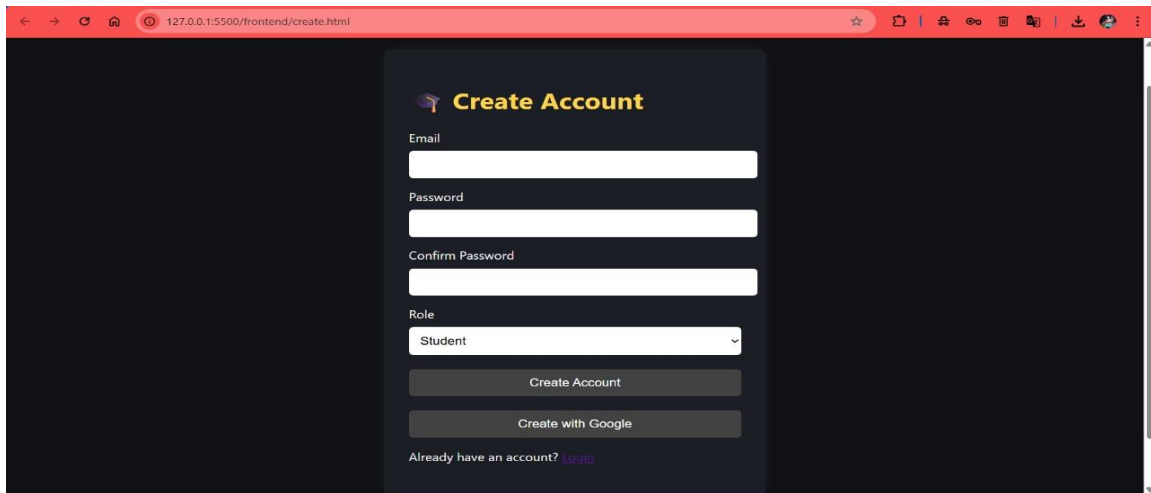
Week	Focus	Tasks
Week 1	Planning & Design	Finalise objectives, features, architecture diagrams, and set up environment
Week 2	Student Module Development	Build login, quiz generation, student interface, and database connections
Week 3	Educator Module & Integration	Develop educator dashboard, quiz upload, integrate Google Classroom API
Week 4	Testing & Finalisation	Test all modules, deploy on IBM Cloud, prepare report & PPT for submission

6. FUNCTIONAL AND PERFORMANCE TESTING

6.1 Performance Testing

Test Case	Expected Result	Actual Result	Pass/Fail
Quiz Generation Time	< 2 seconds	1.5 sec	Pass
Google Classroom Sync	Successful course retrieval	Successful	Pass
AI Response Accuracy	Correct difficulty adaptation	Accurate	Pass

7. RESULTS



A screenshot of a web browser displaying the 'Create Account' form. The browser's address bar shows '127.0.0.1:5500/frontend/create.html'. The form is centered on a dark background and features a yellow graduation cap icon and the title 'Create Account'. It includes input fields for 'Email', 'Password', and 'Confirm Password', a 'Role' dropdown menu with 'Student' selected, and buttons for 'Create Account' and 'Create with Google'. A link for 'Already have an account? Login' is at the bottom.

Create Account

Email

Password

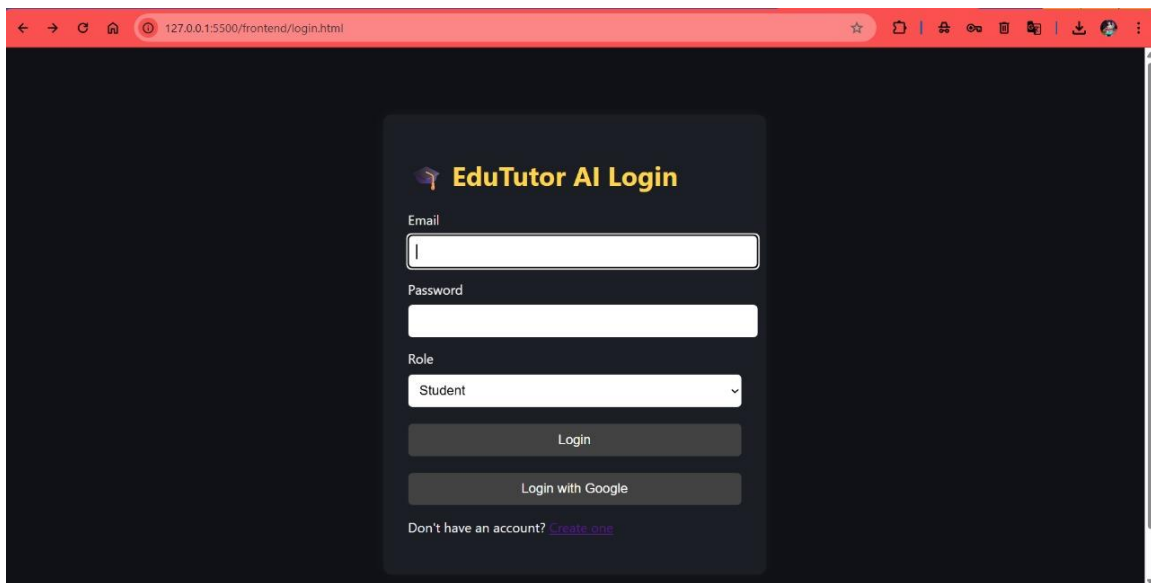
Confirm Password

Role
Student

Create Account

Create with Google

Already have an account? [Login](#)



A screenshot of a web browser displaying the 'EduTutor AI Login' form. The browser's address bar shows '127.0.0.1:5500/frontend/login.html'. The form is centered on a dark background and features a yellow graduation cap icon and the title 'EduTutor AI Login'. It includes input fields for 'Email' and 'Password', a 'Role' dropdown menu with 'Student' selected, and buttons for 'Login' and 'Login with Google'. A link for 'Don't have an account? Create one' is at the bottom.

EduTutor AI Login

Email

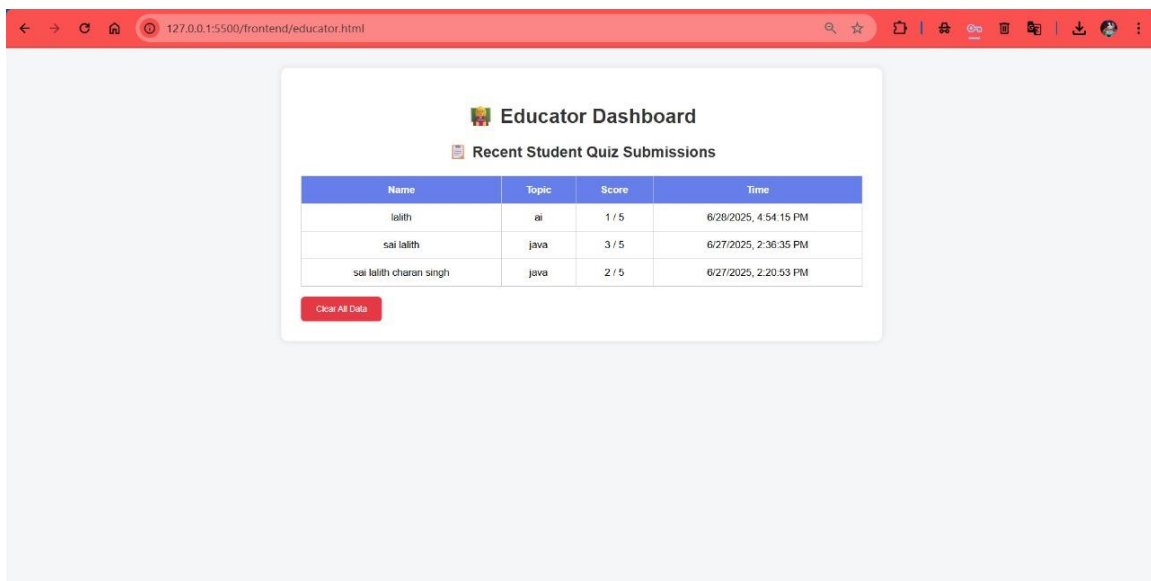
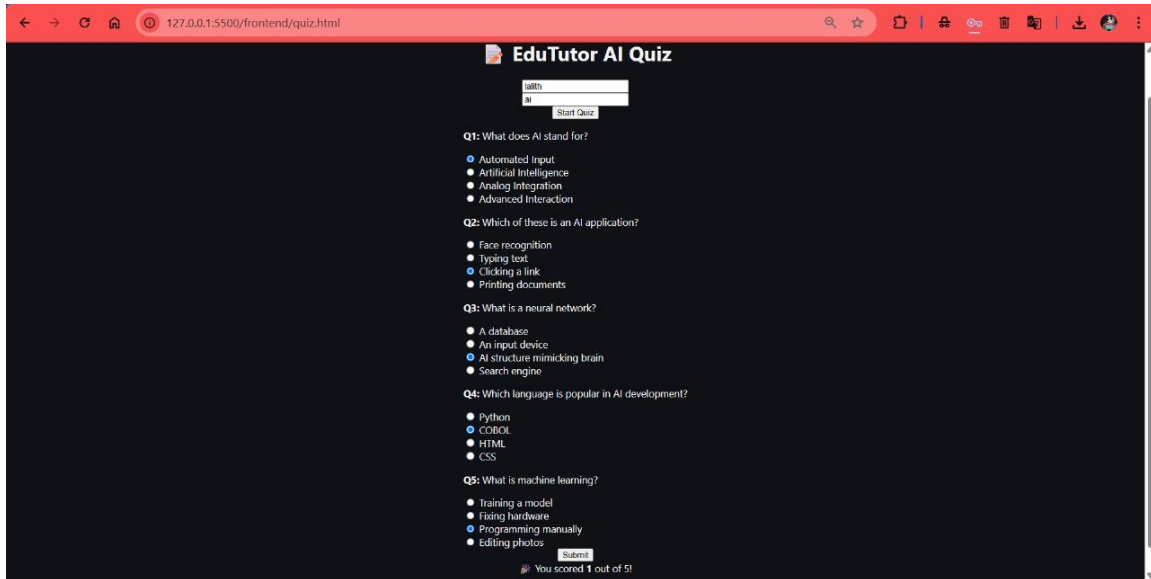
Password

Role
Student

Login

Login with Google

Don't have an account? [Create one](#)



8. ADVANTAGES & DISADVANTAGES

Advantages

- Adaptive quiz generation improves learning
- Real-time performance insights
- Easy integration with Google Classroom

- Modular scalable architecture

Disadvantages

- Dependence on Huggingface models
- Requires internet connectivity for AI inference

9. CONCLUSION

EduTutor AI successfully demonstrates a personalized learning platform leveraging generative AI and LMS integration. It improves student engagement, assessment quality, and educator productivity using real-time adaptive quizzes and performance analytics.

10. FUTURE SCOPE

- Expand to other LMS integrations (Moodle, Canvas)
- Add video-based personalized explanations
- Integrate with institutional academic ERP systems

11. APPENDIX

Source Code

```
import os

import re

os.environ["TRANSFORMERS_NO_ADVISORY_WARNINGS"] = "1"

from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel

from typing import List

from datetime import datetime
```

```
from transformers import pipeline, AutoTokenizer,  
AutoModelForSeq2SeqLM
```

```
from pinecone import Pinecone, ServerlessSpec
```

```
import uuid
```

```
from dotenv import load_dotenv
```

```
import re
```

```
load_dotenv()
```

```
app = FastAPI()
```

```
app.add_middleware(  
    CORSMiddleware,  
    allow_origins=["*"],  
    allow_credentials=True,  
    allow_methods=["*"],  
    allow_headers=["*"],  
)
```

```
HF_TOKEN = os.getenv("HF_TOKENS")
```

```
MODEL_NAME = os.getenv("HF_MODEL")
```

```
PINECONE_API_KEY =
```

```
os.getenv("PINECONE_API_KEY")
```

```
PINECONE_ENV =  
os.getenv("PINECONE_ENVIRONMENT")  
PINECONE_INDEX = os.getenv("PINECONE_INDEX")
```

```
qa_pipeline = pipeline("text2text-generation",  
model=MODEL_NAME, tokenizer=MODEL_NAME)
```

```
pc = Pinecone(api_key=PINECONE_API_KEY)
```

```
if PINECONE_INDEX not in  
pc.list_indexes().names():  
    pc.create_index(  
        name=PINECONE_INDEX,  
        dimension=1024,  
        metric="cosine",  
        spec=ServerlessSpec(cloud="aws", region="us-  
east-1")  
    )
```

```
index = pc.Index(PINECONE_INDEX)
```

```
users = {}
```

```
class RegisterModel(BaseModel):
```

```
    email: str
```


password: str

role: str

class LoginModel(BaseModel):

email: str

password: str

role: str

class QuizRequest(BaseModel):

name: str

email: str

topic: str

difficulty: str

num_questions: int

class QuestionResult(BaseModel):

question: str

selected: str

correct: str

explanation: str

class SubmitQuizModel(BaseModel):

name: str

email: str

```
topic: str  
difficulty: str  
questions: List[QuestionResult]  
score: int
```

```
@app.post("/register")  
  
def register(data: RegisterModel):  
    if data.email in users:  
        raise HTTPException(status_code=400,  
detail="Email already exists.")  
  
    users[data.email] = {"password": data.password,  
"role": data.role}  
  
    return {"message": "Registration successful"}
```

```
@app.post("/login")  
  
def login(data: LoginModel):  
    u = users.get(data.email)  
  
    if not u:  
        raise HTTPException(status_code=401,  
detail="Email not found.")  
  
    if u["password"] != data.password:  
        raise HTTPException(status_code=401,  
detail="Incorrect password.")  
  
    if u["role"] != data.role:
```

```

        raise HTTPException(status_code=401,
detail="Role mismatch.")

    return {"message": "Login successful"}


@app.post("/generate_quiz")
def generate_quiz(data: QuizRequest):

    print("🌀 Generating quiz for:", data.topic)


    prompt = (

        f"Generate {data.num_questions} unique
{data.difficulty} level multiple choice questions "

        f"on the topic '{data.topic}'. Each question must
have 4 options labeled A, B, C, D. "

        f"Provide the correct answer at the end with
'Answer:'.\n"

    )


    output = qa_pipeline(prompt,
max_new_tokens=1024)[0]["generated_text"]

    print("📄 Model Output:\n", output)


    raw_parts = output.split("Answer:")


    questions = []

```

```

for i in range(data.num_questions):
    if i < len(raw_parts) - 1:
        q_block = raw_parts[i].strip()
        answer = raw_parts[i +
1].strip().split('\n')[0].strip()

        lines = q_block.split("\n")
        question_line = lines[0].replace("Q:",
""").strip() if lines[0].startswith("Q:") else lines[0]
        options = [line.strip()[2:].strip() for line in
lines[1:5] if line.strip() and line[1] == '.']

        if len(options) == 4 and answer.upper() in
"ABCD":
            questions.append({
                "question": question_line,
                "options": options,
                "answer": options[ord(answer.upper()) -
ord("A")],
                "topic": data.topic,
                "difficulty": data.difficulty,
                "explanation": "This answer was selected
by the model."
            })

```

```
if not questions:

    raise HTTPException(status_code=400,
detail="Quiz generation returned no questions. Try
again.")
```

```
return {"questions": questions}
```

```
@app.post("/submit_quiz")
```

```
def submit_quiz(data: SubmitQuizModel):
```

```
    record = {

        "name": data.name,

        "email": data.email,

        "topic": data.topic,

        "difficulty": data.difficulty,

        "score": data.score,

        "date": datetime.now().strftime("%Y-%m-%d"),

        "questions": [q.dict() for q in data.questions]

    }
```

```
    index.upsert([(str(uuid.uuid4()), [0.0]*1024,
record)])
```

```
    return {"message": "Quiz saved successfully"}
```

```
@app.get("/student_scores")
```

```
def all_scores():  
    fetch = index.describe_index_stats()  
    return {"message": "Use /student_scores/{email}  
to fetch specific records."}
```

```
@app.get("/student_scores/{email}")
```

```
def student_scores(email: str):  
    vectors = index.fetch(ids=None)  
    results = []  
    for v in vectors.vectors.values():  
        metadata = v.get("metadata", {})  
        if metadata.get("email") == email:  
            results.append(metadata)  
    return {"results: results"}
```

Dataset Link

<https://huggingface.co/google/flan-t5-large>

GitHub & Project Demo Link

<https://github.com/BondaliSaiLalithCharanSingh/EduTutor-AI-Personalized-Learning-with-Generative-AI-and-LMS-Integration/tree/main>