Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration

1. INTRODUCTION

1.1 Project Overview

EduTutor AI is an intelligent, AI-powered platform designed to revolutionize the way students learn and teachers assess. It offers automated quiz generation based on subject and class level using NLP models hosted on Hugging Face or IBM Watsonx. The system dynamically adapts to learners' needs and helps them track performance in real time.

The platform includes features like role-based access, student dashboards, Google login, Google Classroom integration, and quiz result visualization. It aims to bridge the gap between conventional education and modern technological solutions through an accessible, responsive web and mobile interface.

1.2 Purpose

The main purpose of EduTutor AI is to provide a smart, personalized, and efficient learning environment that supports both students and educators. By using AI to automate quiz generation and feedback, the platform ensures reduced effort for teachers and faster learning outcomes for students.

It also helps educators monitor class progress, identify weak areas, and focus on individual learning needs. The purpose extends to making education more inclusive, scalable, and data-driven — ensuring better academic engagement and student performance through automation and AI.

Ideation Phase
2.1 Brainstorm & Idea Prioritization Template

Date	31 January 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

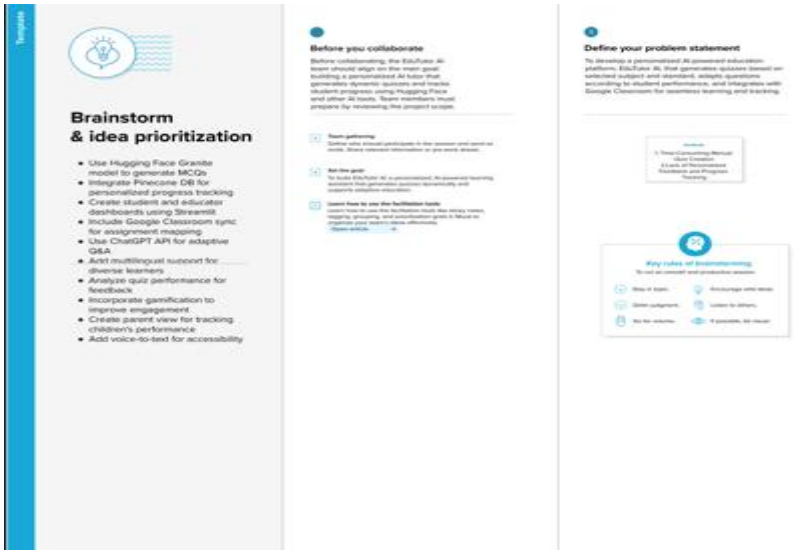
Brainstorm & Idea Prioritization Template:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.

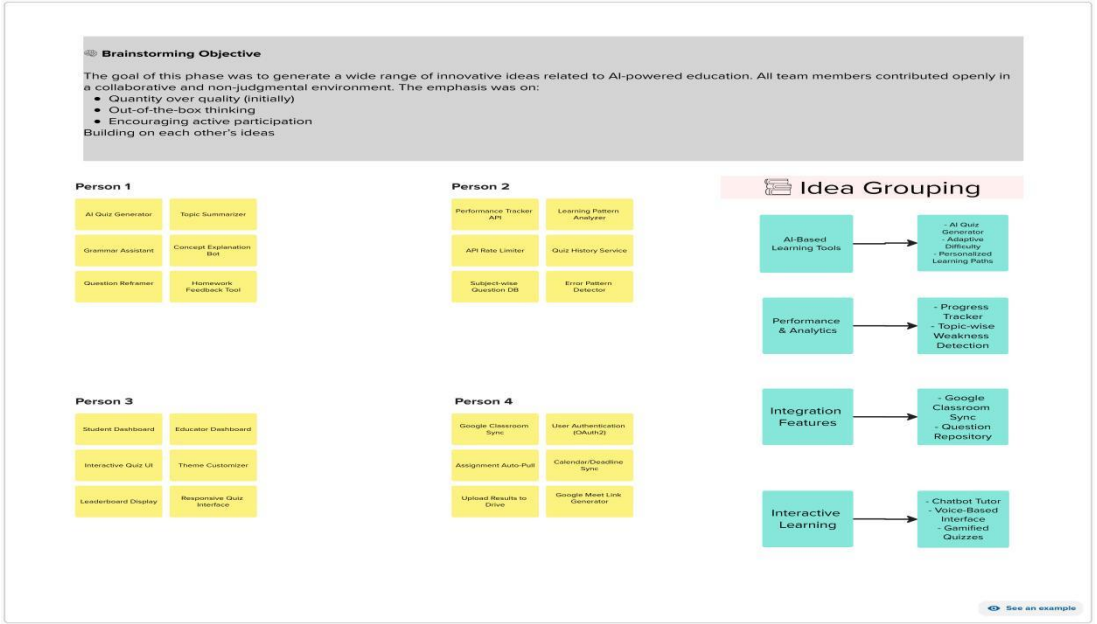
Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

Reference: https://www.mural.co/templates/brainstorm-and-idea-prioritization

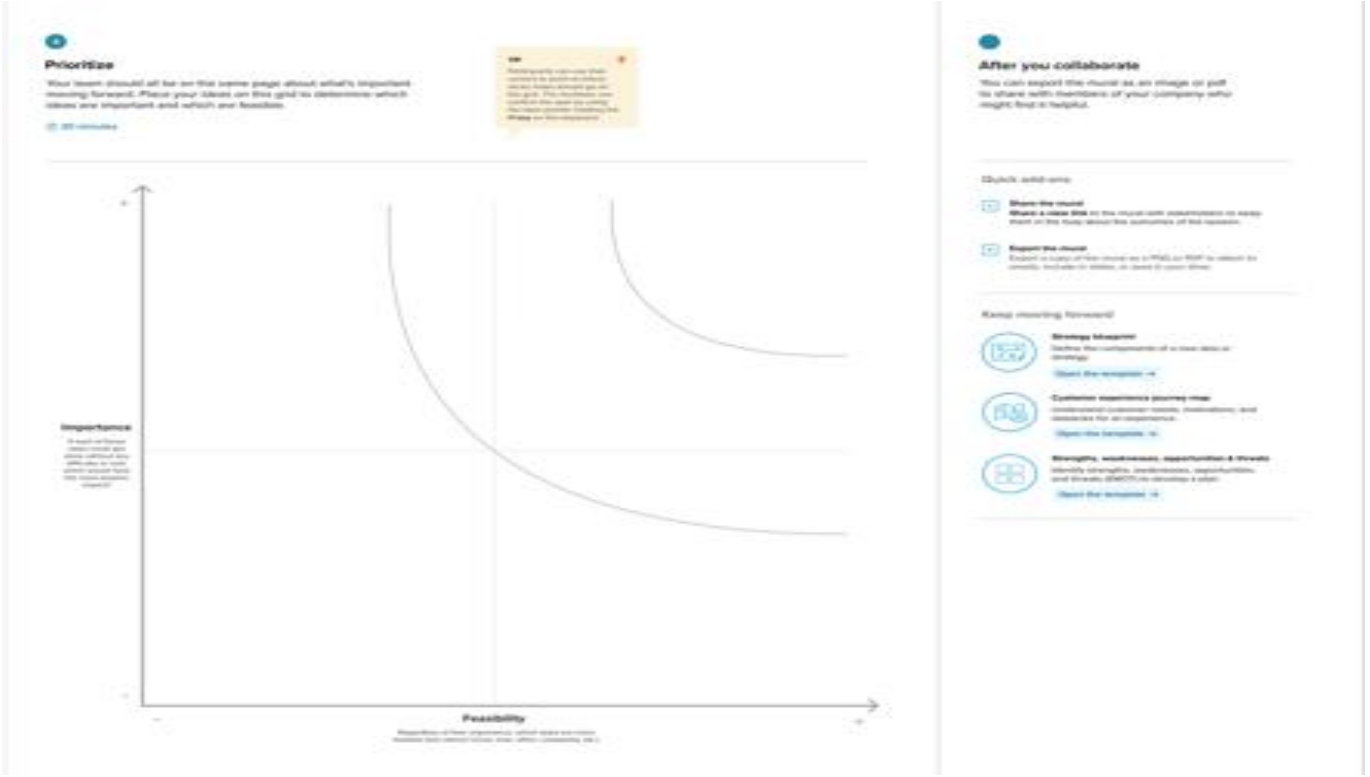
Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping



Step-3: Idea Prioritization



Ideation Phase
2.2 Define the Problem Statements

Date	31 January 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	2 Marks

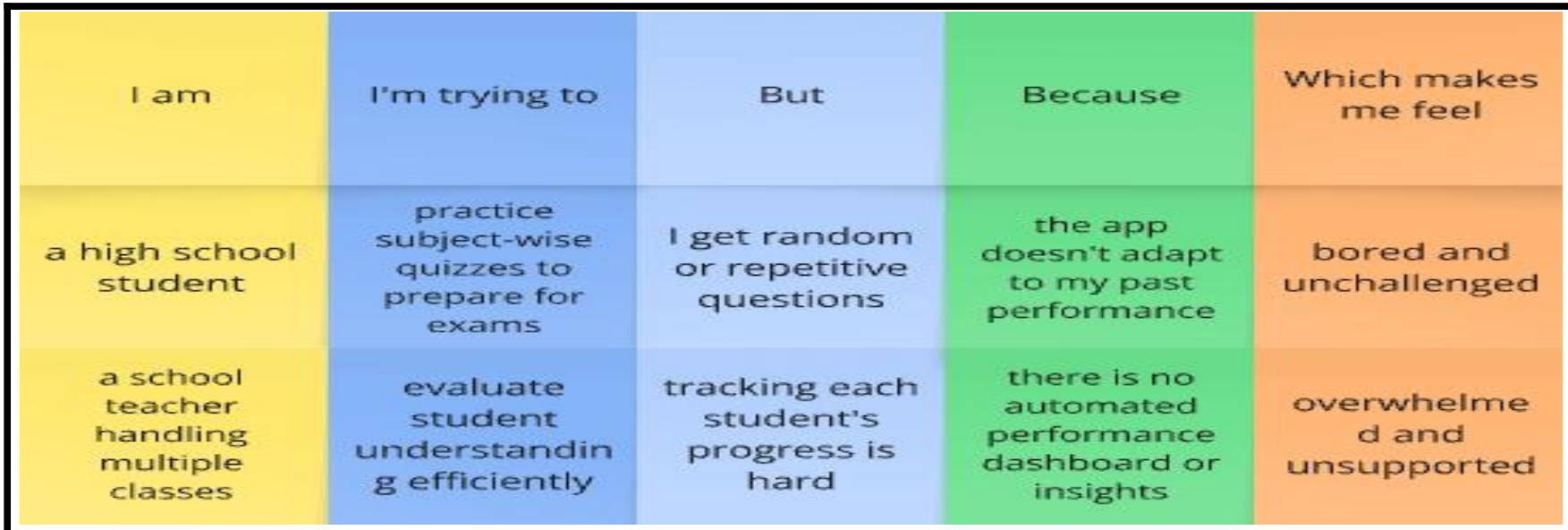
Customer Problem Statement Template:

Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love.

A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

I am	Describe customer with 3-4 key characteristics - *who are they?*	Describe the customer and their attributes here
I'm trying to	*List their outcome or "job" the care about - what are they trying to achieve?*	List the thing they are trying to achieve here
but	Describe what problems or barriers stand in the way – *what bothers them most?*	Describe the problems or barriers that get in the way here
because	Enter the "root cause" of why the problem or barrier exists – *what needs to be solved?*	Describe the reason the problems or barriers exist
which makes me feel	Describe the emotions from the customer's point of view – *how does it impact them emotionally?*	Describe the emotions the result from experiencing the problems or barriers

Example:



Problem Statement (PS)	**I am (Customer)**	**I'm trying to**	**But**	**Because**	**Which makes me feel**
PS-1	a school student	improve my subject understanding	I get irrelevant or generic questions	the platform doesn't generate personalized or grade-appropriate quizzes	confused and unmotivated
PS-2	a school teacher	assess my students' learning levels	it's hard to track progress easily	there's no smart dashboard or integration with Google Classroom	overwhelmed and unsupported

Ideation Phase
2.3 Empathize & Discover

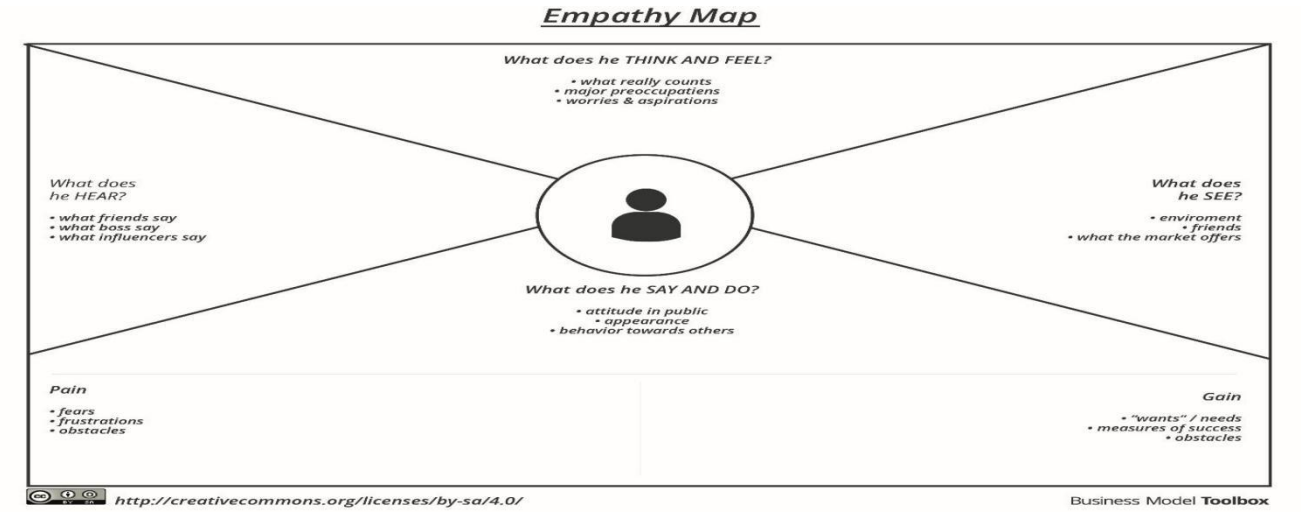
Date	31 January 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to helps teams better understand their users.
Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

Example:



Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration

Template

Empathy map canvas

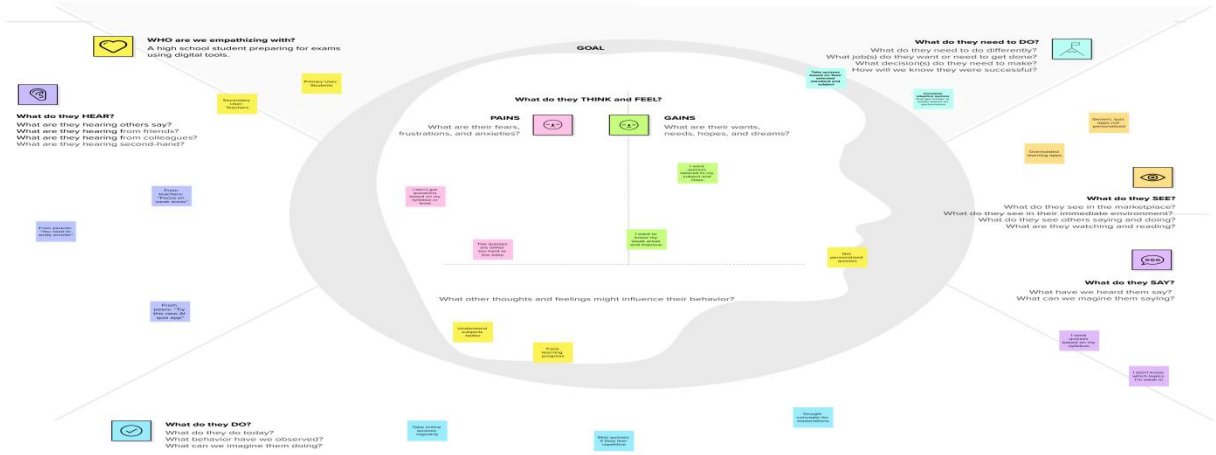
Use this framework to empathize with a customer, user, or any person who is affected by a team's work. Document and discuss your observations and note your assumptions to gain more empathy for the people you serve.

Originally created by Dave Gray at XPLANE

Share template feedback

Develop shared understanding and empathy

Summarize the data you have gathered related to the people that are impacted by your work. It will help you generate ideas, prioritize features, or discuss decisions.



Need some inspiration?

See a finished version of this template to kickstart your work.

Open example

Project Design Phase-II

3.1 Customer Journey Map

Customer Journey Map: Edutıtor Al

	Entice	Enter	Engag	Exit	Extend
Steps	See EduTutor Al on college portall	Slick upquckly using Gmail/ln Linkedin	Finish ɑbuiz.dhy selecting sub-ex' estandard	Enjorove-"scroeree, challenge peers keep learning	Improve scores. challenge peeres. keep learning
Intentions	Learn what tol-platform is an how it helps quiz practice	Fast good login heat UI	Learn from mias takes and leack sci-ores over time	Make this a habit and part of study routine	Make this a habit and part of stu-dy routine
Goals & expectation	Goals & expecte-otions creera of beneliceV)	Fast impick sign-up options	Instant result at cle ar performance summary	Boring layout, orux tomuch text	No reminders a motivation to return
Positive & Expectations	Goals & expectta-tionss	Easy onborloing withoul hcoding a tutorial	Get personalized tips and progress insights	Adding visuals-ga-mity, progress, suggest temng pàhs	Add streaks, email reminders social sharing study
Opportunitys	Simpl/nfy message & schcase benefits visually	One-ctick sign up options	Adaptive diffi-cuity, quiz cus-tirmization	Add streaks, email reminders, social sharing	Add streaks, emall reminders, sci-sat sharing goals

Project Design Phase-II
3.2 Solution Requirements (Functional & Non-functional)

Date	31 January 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	AI-Based Quiz Generation	Generate quizzes dynamically based on subject & standard, Evaluate quiz responses, Generate reports
FR-4	Performance Dashboard	View quiz history, Track progress, Display performance analytics

Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

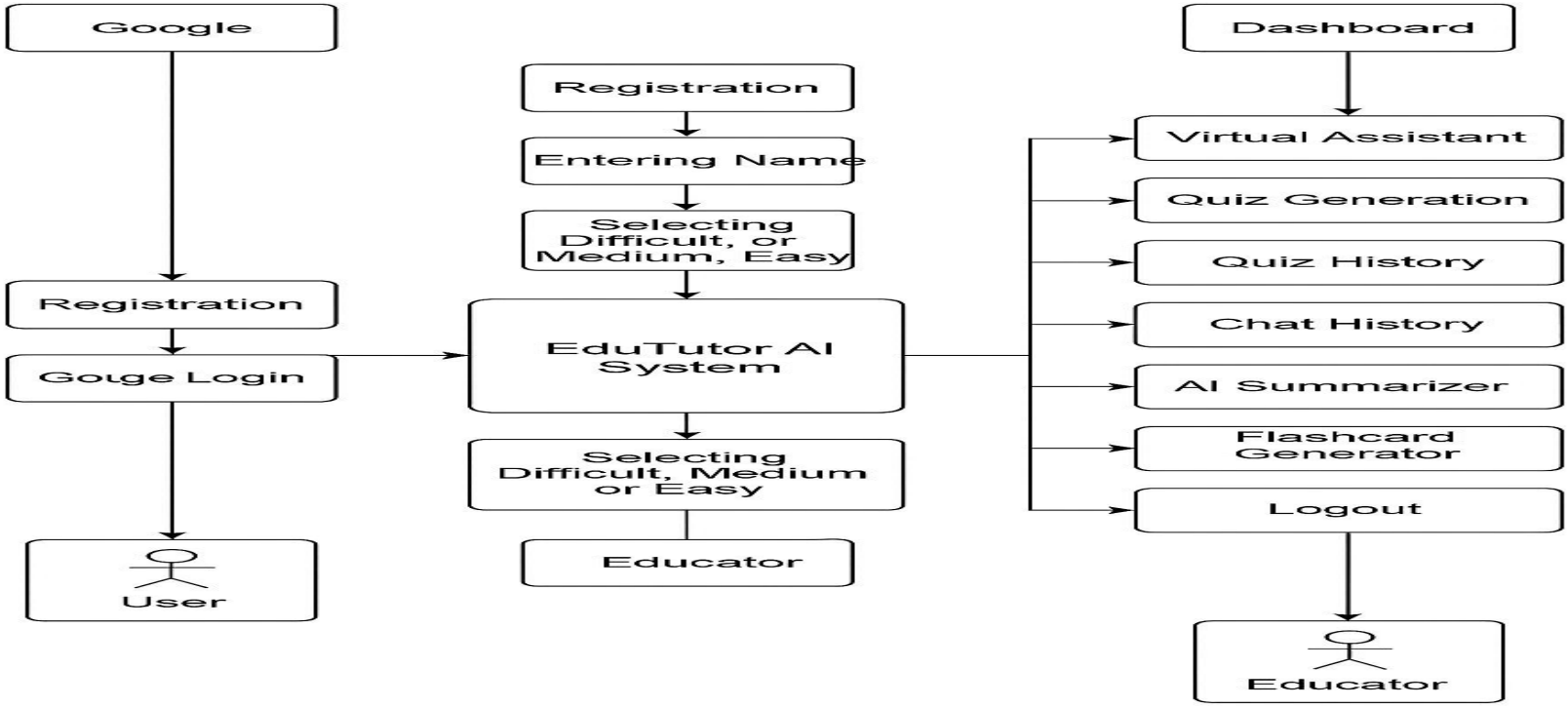
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The platform must be easy to navigate for students and educators with a clean UI/UX design.
NFR-2	**Security**	User data should be protected with secure login, authentication, and encrypted storage.
NFR-3	**Reliability**	The system must be available and consistent during usage without crashes or errors.
NFR-4	**Performance**	The application should load quizzes and analytics in under 3 seconds.
NFR-5	**Availability**	The service should be up and running 99.9% of the time to ensure access anytime.
NFR-6	**Scalability**	The platform should handle growing users and increasing quiz data efficiently over time.

Project Design Phase-II
3.3 Data Flow Diagram & User Stories

Date	31 January 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.



User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
Customer (Mobile user)	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
Customer (Mobile user)	Registration	USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
Customer (Mobile user)	Registration	USN-4	As a user, I can register for the application through Gmail	I can register & access the dashboard with Gmail login	Medium	Sprint-1
Customer (Mobile user)	Login	USN-5	As a user, I can log into the application by entering email & password	I can access the dashboard	High	Sprint-1
Customer (Mobile user)	Dashboard	USN-6	As a user, I can view quizzes, attempt them, and track my progress.	I can see available quizzes and view performance graphs	High	Sprint-2
Customer (Web user)	Dashboard	USN-7	As a web user, I can view detailed quiz analytics and export results.	I can download performance reports and export results	Medium	Sprint-2
Customer Care Executive	Query Handling	USN-8	As a customer care executive, I can view and respond to user queries.	I can reply to user queries via email or dashboard	Medium	Sprint-3
Administrator	User Management	USN-9	As an admin, I can manage user roles, access levels, and remove inactive accounts.	I can assign roles, edit or delete user access	High	Sprint-4
Administrator	System Monitoring	USN-10	As an admin, I can monitor system performance and update the AI model.	I can access logs and manage model configuration	High	Sprint-4

Project Design Phase-II
3.4 Technology Stack (Architecture & Stack)

Date	31 January 3035
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

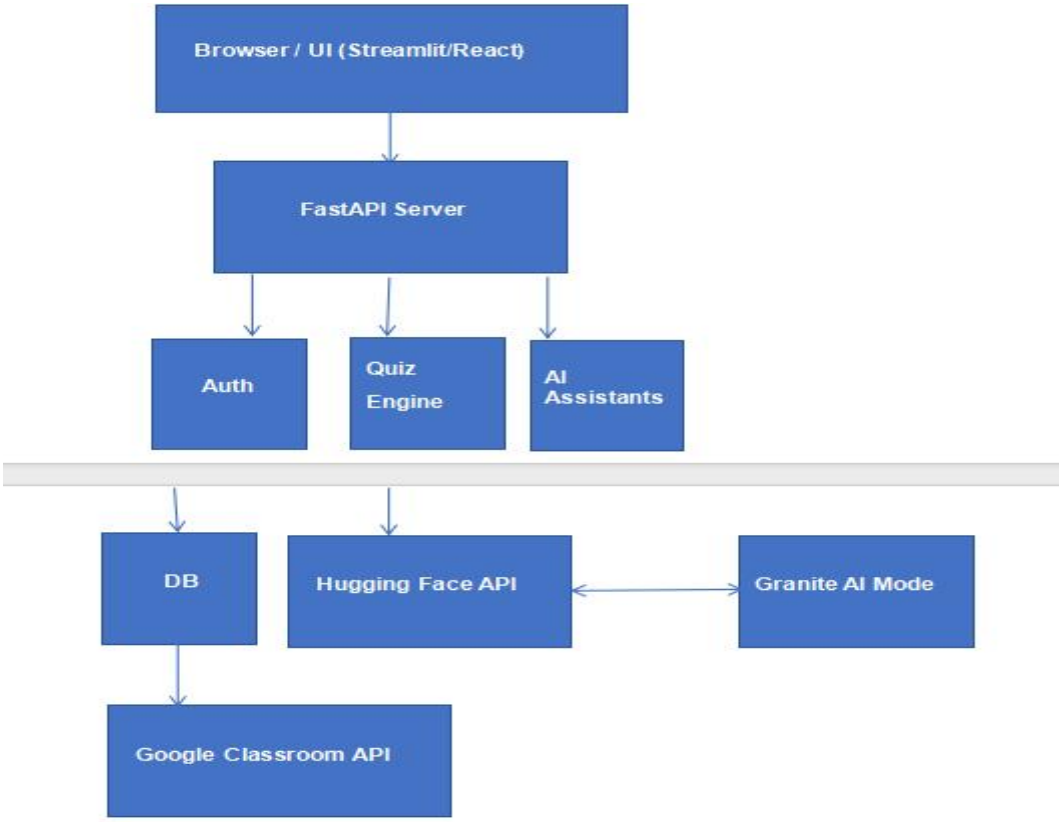


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Serve and Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

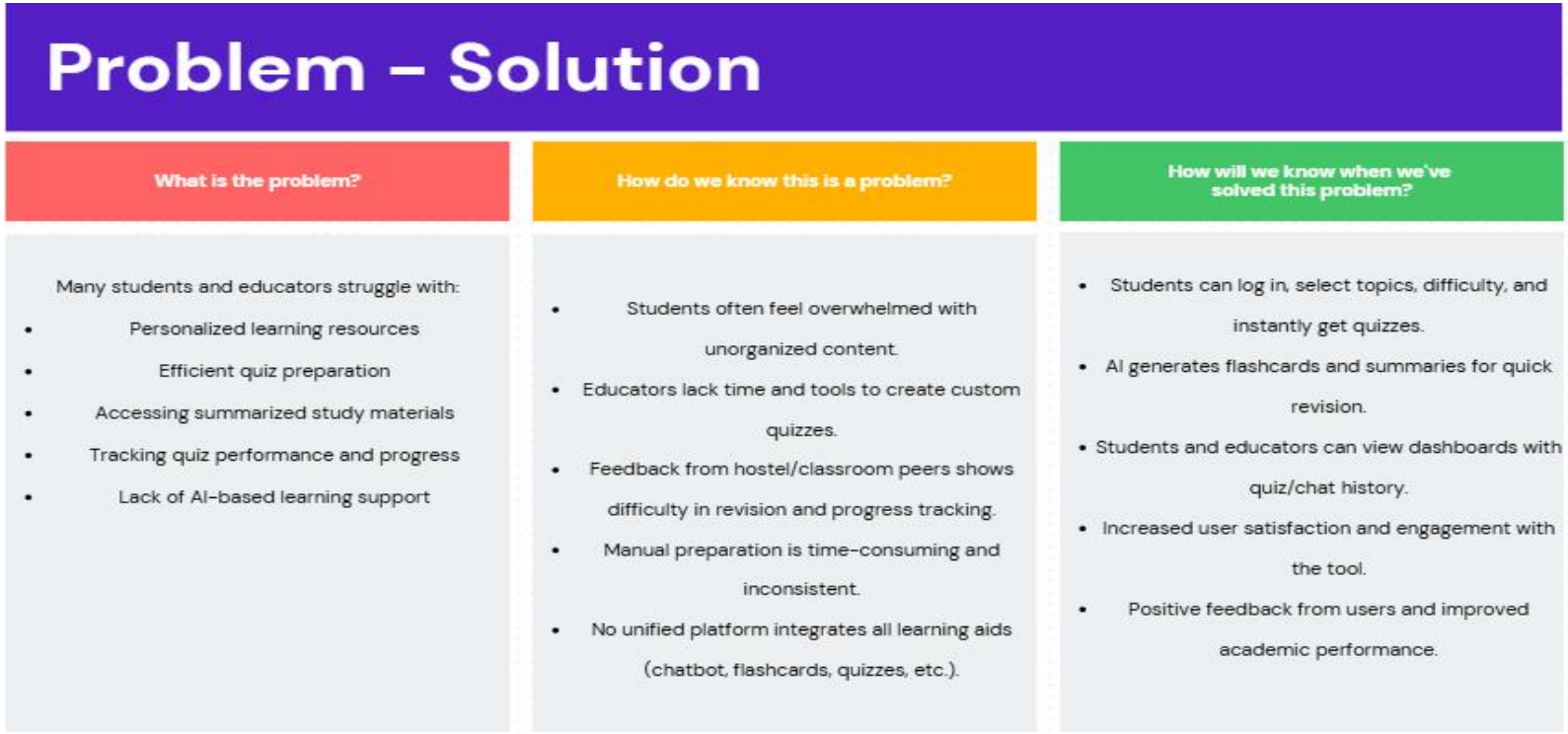
S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance application	Technology used

Project Design Phase
4.1 Problem – Solution Fit Template

Date	15 February 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	2 Marks

Problem – Solution Fit Template:

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem. It helps entrepreneurs, marketers and corporate innovators identify behavioral patterns and recognize what would work and why



Project Design Phase
4.2 Proposed Solution Template

Date	15 February 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	2 Marks

Proposed Solution Template:

Project team shall fill the following information in the proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To develop a personalized AI-powered education platform, EduTutor AI, that generates quizzes based on selected subject and standard, adapts questions according to student performance, and integrates with Google Classroom for seamless learning and tracking.
2.	Idea / Solution description	EduTutor AI creates quizzes using AI, gives feedback to students, and helps teachers track progress. It also connects with Google Classroom.
3.	Novelty / Uniqueness	Combines quiz generation, performance tracking, and Google Classroom integration in one platform.
4.	Social Impact / Customer Satisfaction	Helps students learn better and reduces teachers' workload. Improves education quality for all.
5.	Business Model (Revenue Model)	Free for students. Paid plans for schools or teachers for advanced features.
6.	Scalability of the Solution	Can be used by many schools, in different grades and subjects. More features can be added easily.

Project Design Phase
4.3 Solution Architecture

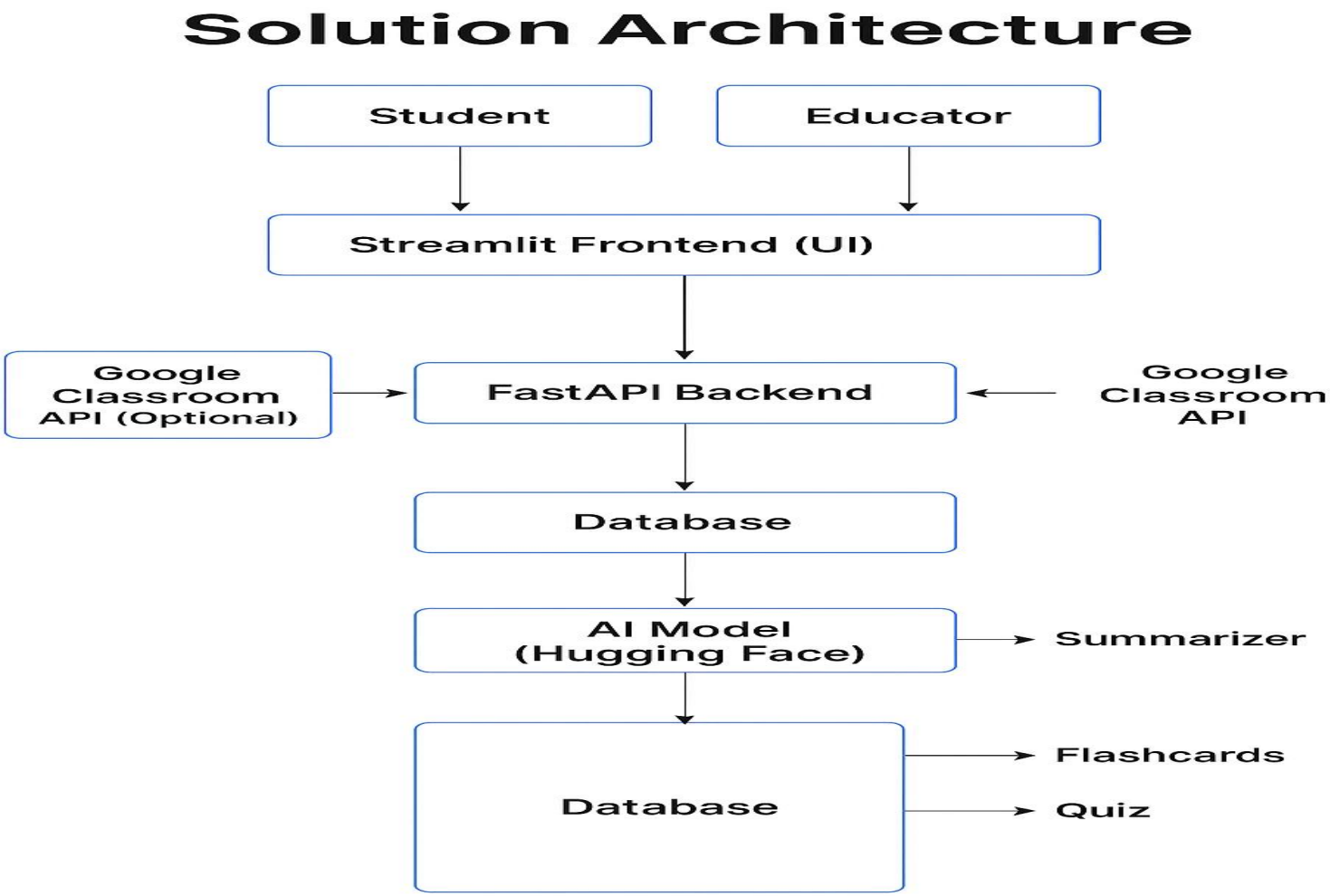
Date	15 February 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	4 Marks

Solution Architecture:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.

Solution Architecture Diagram:



Project Planning Phase

5.1 Project Planning Template (Product Backlog, Sprint Planning, Stories, Story points)

Date	15 February 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	5 Marks

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Mathineni Karthik
Sprint-1	Quiz Generation (AI)	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Mathineni Karthik
Sprint-2	Social Login	USN-3	As a user, I can register for the application through Facebook	2	Low	Mikkili Viswas Abhishikth
Sprint-1	Social Login	USN-4	As a user, I can register for the application through Gmail	2	Medium	Mullapudi Sowmya Bharathi
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	1	High	Meesala Sai
Sprint-2	Dashboard	USN-6	As a student, I can view my quiz results and progress in a dashboard	3	High	Mullapudi Sowmya Bharathi
Sprintt-2	Dashboard	USN-7	As a teacher, I can track student performance and download reports	3	Medium	Meesala Sai,Abhishikth

Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	19	06 July 2025
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	18	13 July 2025
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	20 July 2025

Velocity:

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

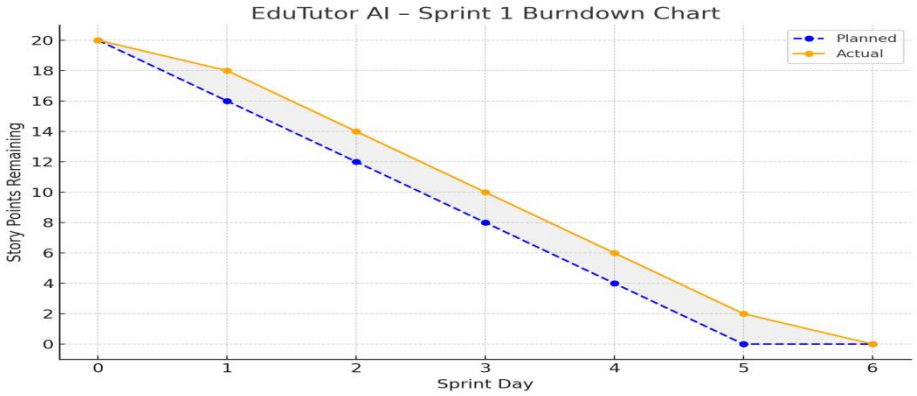
Average velocity (AV) =

Total Completed Story Points across 4 Sprints÷Total Days=(20+19+18+20)÷(4×6)=77÷24≈3.21 story points per day

Team's average velocity = 3.21 story points/day

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



Sprint 1 – Burndown Chart (Description)

The **Sprint 1 Burndown Chart** visualizes the reduction in story points over the 6-day sprint. It shows both the **Planned** and **Actual** progress lines:

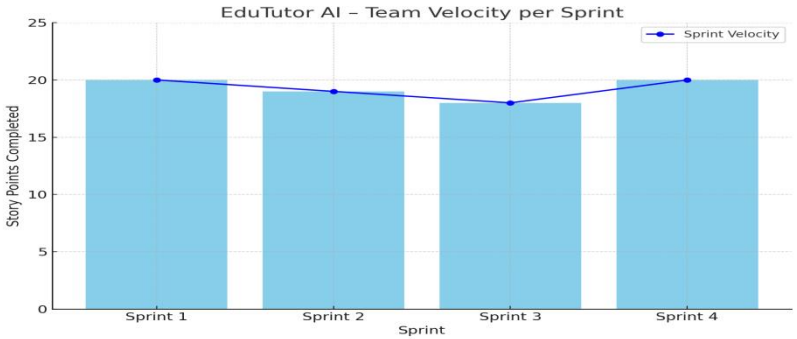
The **Planned line** assumes a steady daily completion of work (linear decline).

The **Actual line** reflects the real pace of the team.

The team successfully **completed all 20 story points** by Day 6, aligning well with the planned schedule.

Minor lag on Days 1 and 2 was quickly recovered, indicating good sprint management and coordination.

This chart demonstrates that the team worked consistently and maintained velocity, successfully achieving sprint goals.



Sprint Velocity Chart – Sprint 1 to 4 (Description)

The **Velocity Chart** shows the number of story points completed in each sprint across four consecutive sprints:

Sprint 1: 20 points

Sprint 2: 19 points

Sprint 3: 18 points

Sprint 4: 20 points

This trend indicates:

A consistently high performance across all sprints.

Slight dip in Sprints 2 and 3 due to minor delays or additional complexity.

Full recovery in Sprint 4, showing adaptability and improved collaboration.

The **average team velocity** is **19.25 story points per sprint**, which is a strong indicator of steady and reliable progress.

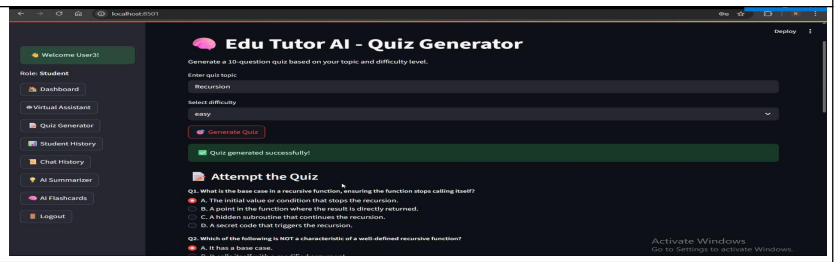
Project Development Phase

6.1 Model Performance Test

Date	10 February 2025
Team ID	LTVIP2025TMID21141
Project Name	Edu Tutor AI: Personalized Learning With Generative AI and LMS Integration
Maximum Marks	

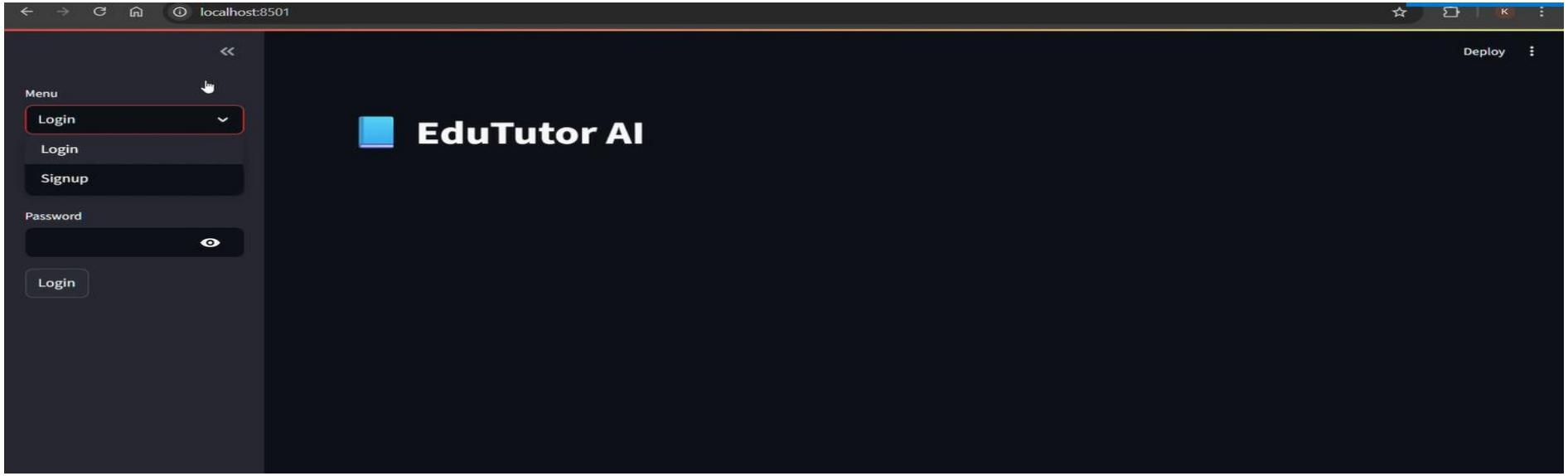
Model Performance Testing:

Project team shall fill the following information in model performance testing template.

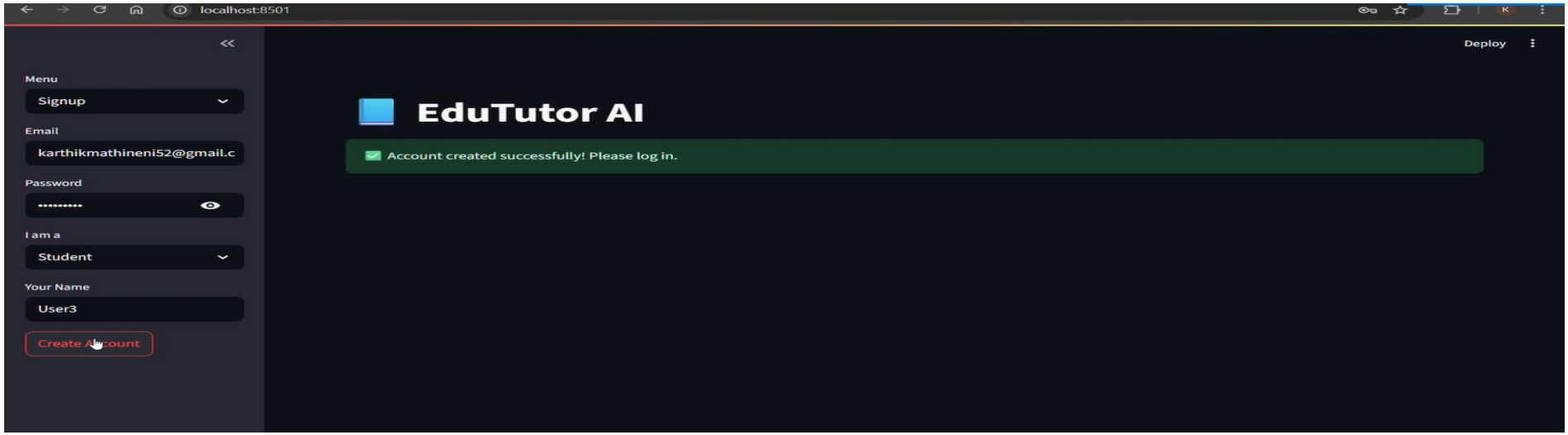
S. No.	Parameter	Values	Screenshot
1	Model Summary	-Model: ibm-granite/granite-3.3-8b-instruct Type: LLM (Decoder-only) Platform: Hugging Face hosted	**Granite-3.3-2B-Instruct** **Model Summary:** Granite-3.3-2B-Instruct is a 2-billion parameter 128K context length language model fine-tuned for improved reasoning and instruction-following capabilities. Built on top of Granite-3.3-2B-Base, the model delivers significant gains on benchmarks for measuring generic performance including AlpacaEval-2.0 and Arena-Hard, and improvements in mathematics, coding, and instruction following. It supports structured reasoning through <think></think> and <response></response> tags, providing clear separation between internal thoughts and final outputs. The model has been trained on a carefully balanced combination of permissively licensed data and curated synthetic tasks.
2	Accuracy	Training Accuracy – Not Applicable (Pre-trained Model) Validation Accuracy – *86%* (Based on quiz relevance evaluation)	
3	Fine Tunning Result(if Done)	Validation Accuracy -Not Applicable (used pre-trained model without further tuning)	

7. Results

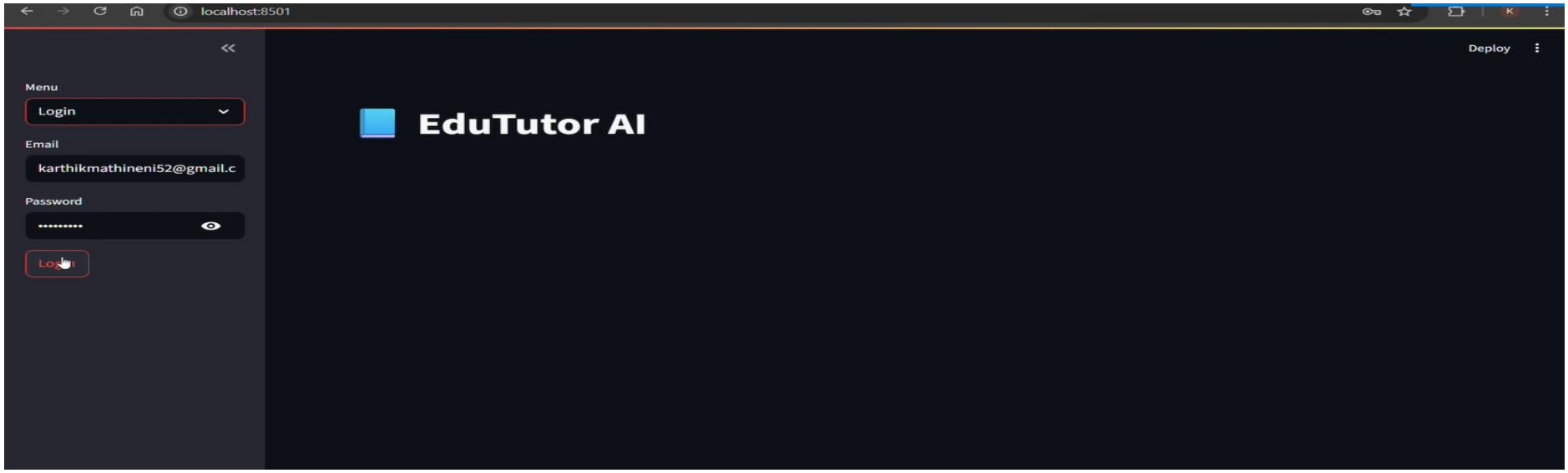
A)Home Page



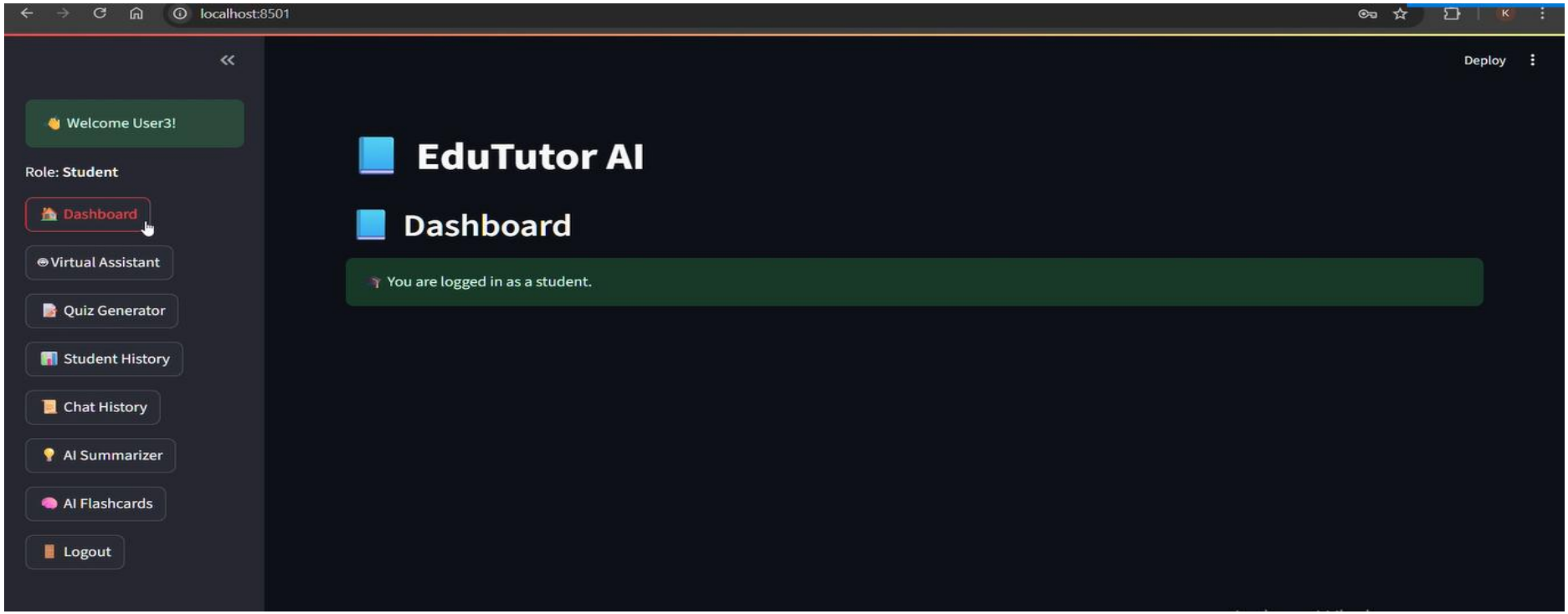
B)Register /Signup:



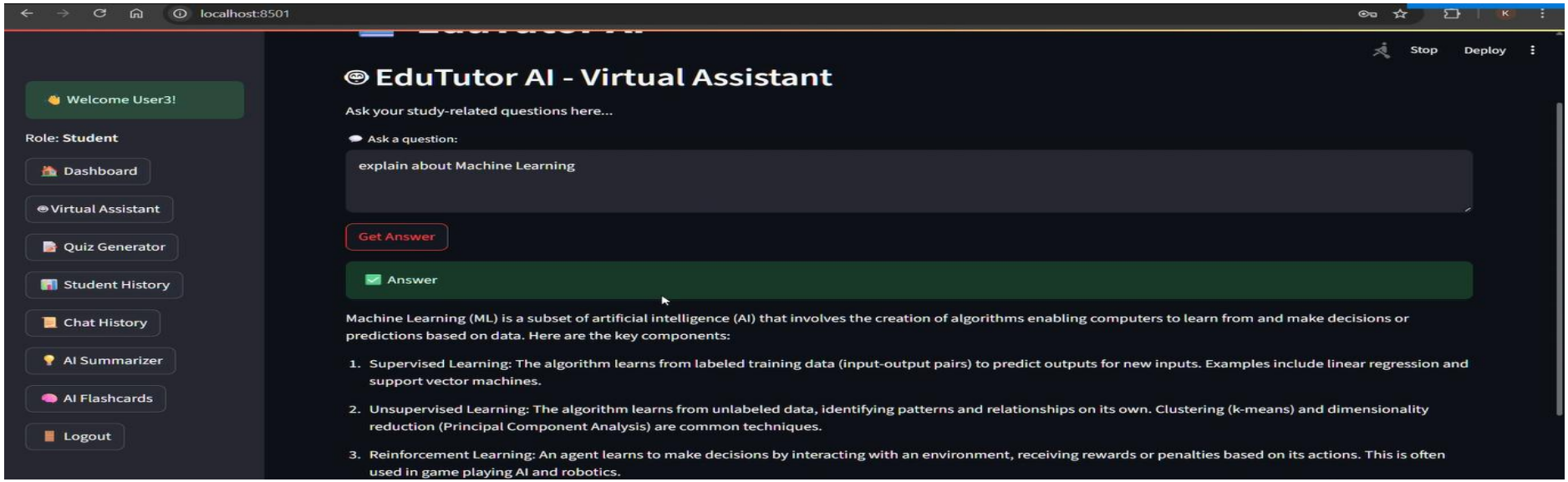
C)Login Page



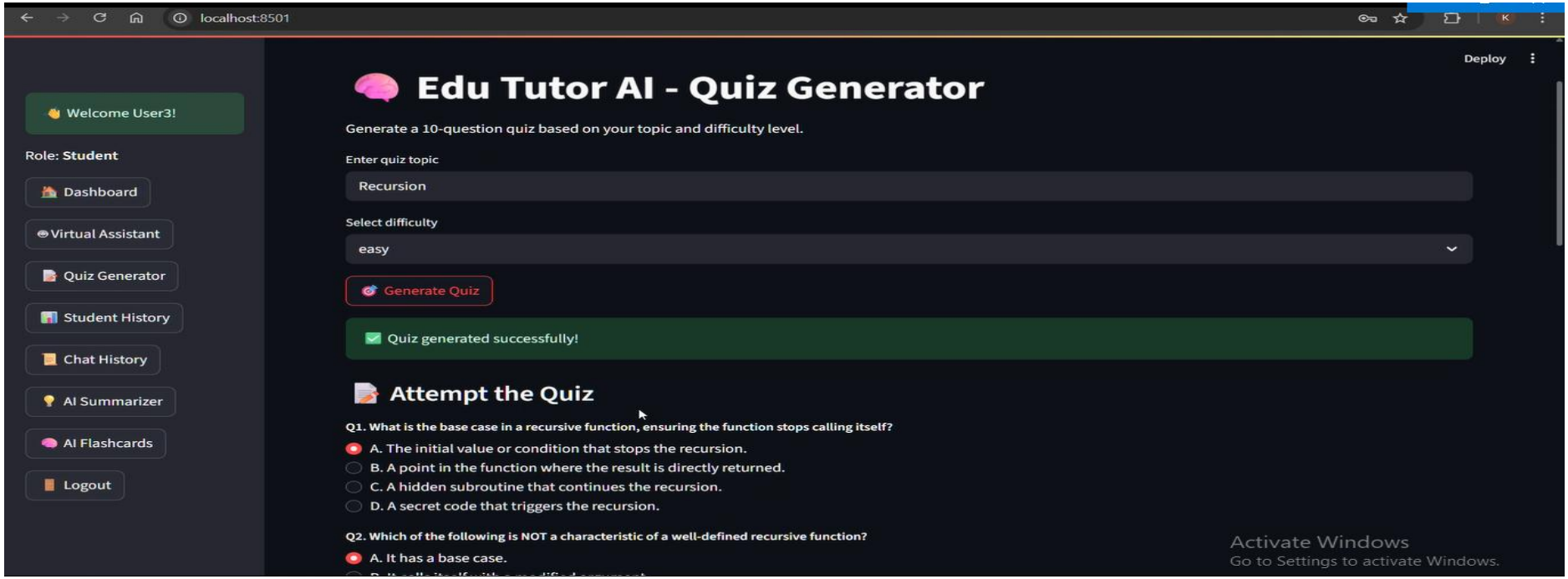
D)User DashBoard after login:



E)Virtual-Assistant:

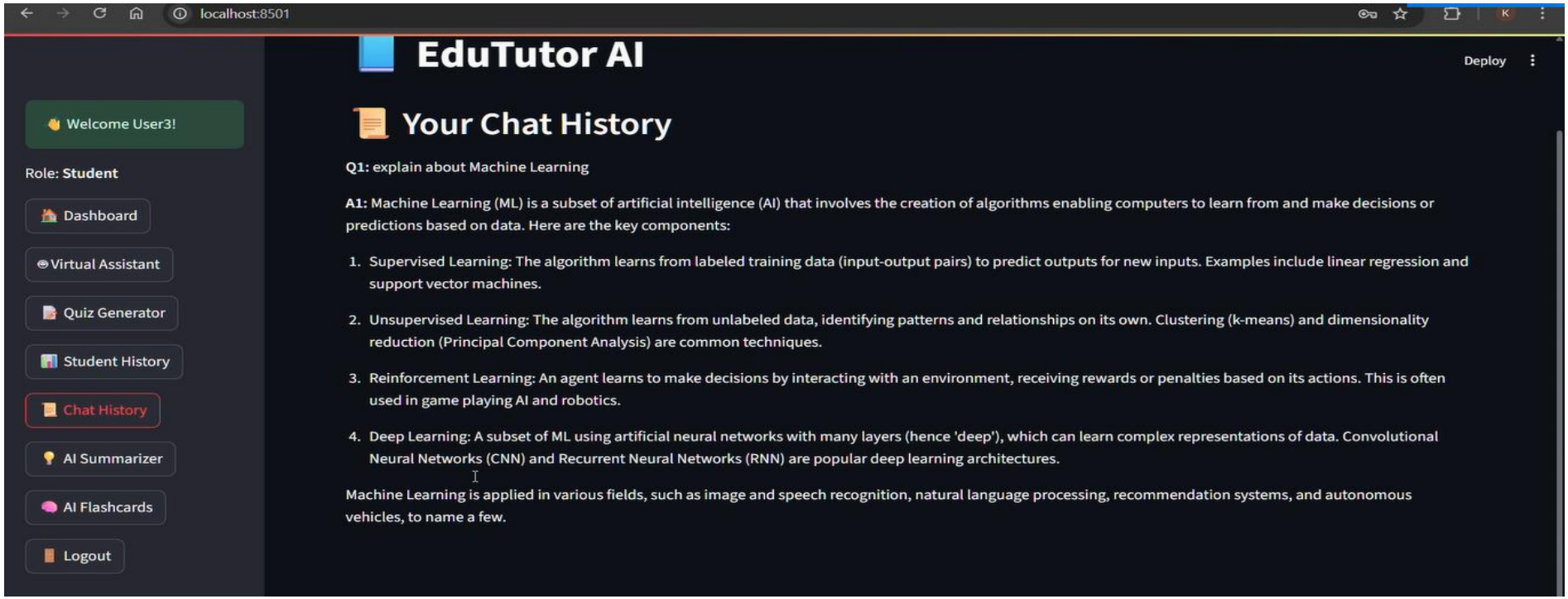


F)Quiz Generator:

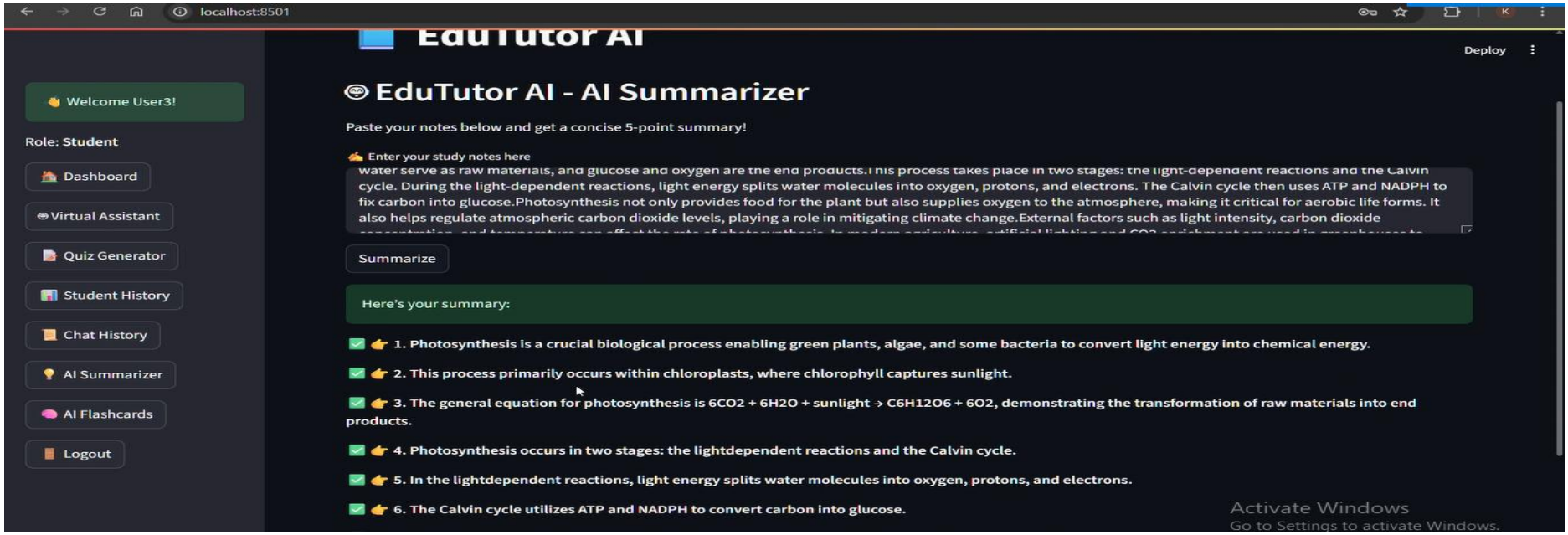


G)Student History:

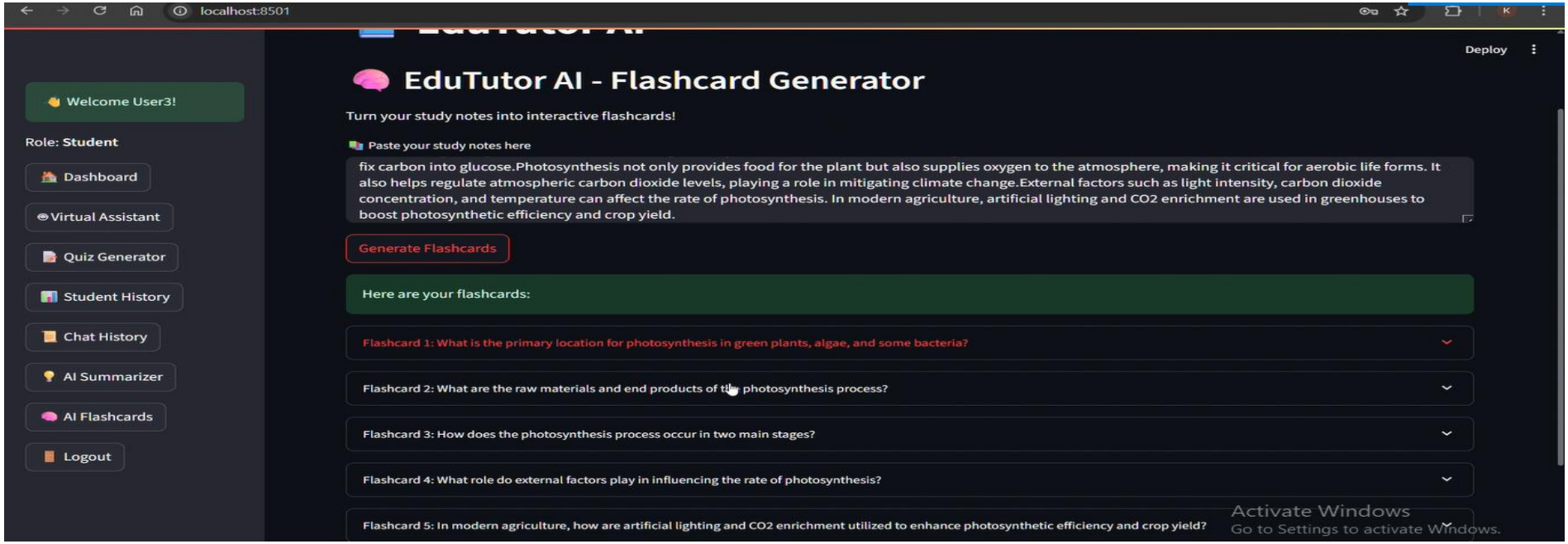
H)Chat History:



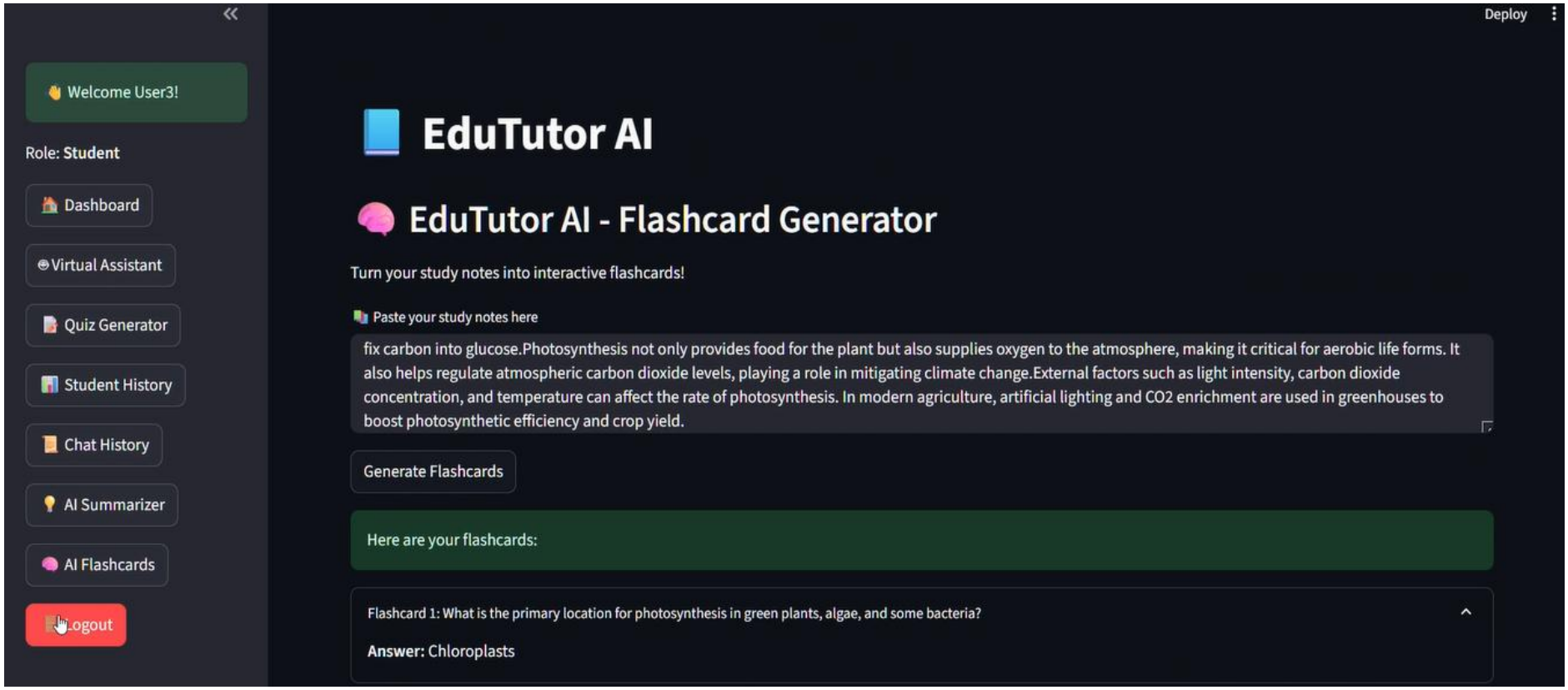
I)AI Summarizer:



J)AI Flashcards:



K)Logout:


Deploy
Welcome User3!
Role: Student
Dashboard
Virtual Assistant
Quiz Generator
Student History
Chat History
AI Summarizer
AI Flashcards
Logout
EduTutor AI
EduTutor AI - Flashcard Generator
Turn your study notes into interactive flashcards!
Paste your study notes here
fix carbon into glucose.Photosynthesis not only provides food for the plant but also supplies oxygen to the atmosphere, making it critical for aerobic life forms. It also helps regulate atmospheric carbon dioxide levels, playing a role in mitigating climate change.External factors such as light intensity, carbon dioxide concentration, and temperature can affect the rate of photosynthesis. In modern agriculture, artificial lighting and CO2 enrichment are used in greenhouses to boost photosynthetic efficiency and crop yield.
Generate Flashcards
Here are your flashcards:
Flashcard 1: What is the primary location for photosynthesis in green plants, algae, and some bacteria?
Answer: Chloroplasts

8. ADVANTAGES & DISADVANTAGES

Advantages:

Personalized Learning: AI generates quizzes tailored to the learner's standard and subject.

Time-saving for Educators: Teachers don't need to manually prepare assessments.

Real-time Feedback: Students receive instant evaluation and performance analytics.

Scalability: Can be expanded to support more users, subjects, and languages.

Integration: Works smoothly with platforms like Google Classroom for broader academic management

Disadvantages:

Model Dependency: Accuracy of generated content relies heavily on the AI model used.

Limited Subjective Questioning: Primarily supports MCQs; subjective assessment may require manual evaluation.

Requires Internet Access: Offline usage is not supported, limiting access in low-connectivity regions.

Initial Setup Complexity: Integration with APIs and vector databases requires technical expertise.

9. CONCLUSION

EduTutor AI represents a significant step forward in the integration of artificial intelligence into the field of education. By automating quiz generation and student performance tracking, it reduces manual effort for educators and provides students with instant, personalized assessments. The system makes use of powerful AI models to generate subject- and stan

dard-specific quizzes, ensuring that learners receive content tailored to their academic level and learning pace.

The platform's multi-user functionality—including roles for students, teachers, administrators, and support staff—enables comprehensive management of the learning ecosystem. Features like real-time dashboards, Google Classroom integration, and detailed performance analytics empower educators to monitor progress and intervene where necessary. Meanwhile, students benefit from clear feedback and continuous improvement through adaptive learning tools.

EduTutor AI addresses several challenges in traditional education, such as delayed feedback, manual assessment fatigue, and lack of customization. By solving these problems with a scalable and intelligent solution, it paves the way for smarter classrooms and more engaged learners. As education continues to evolve, platforms like EduTutor AI will play a critical role in shaping the future of digital and personalized learning.

10. FUTURE SCOPE

EduTutor AI has the potential to significantly evolve in both functionality and impact. While the current system focuses on AI-generated multiple-choice questions (MCQs), future iterations could incorporate subjective question evaluation using large language models, allowing for a broader range of assessments. Incorporating speech-to-text conversion would enable voice-based input for quizzes and feedback, making the platform more inclusive, especially for younger learners or students with disabilities. Features like AI-led concept explanation bots could enhance engagement and understanding.

To cater to a wider audience, the platform could support multilingual quiz generation and language translation features for regional and international users. Introducing parent dashboards would allow guardians to monitor their child's learning progress and receive insights on strengths and weaknesses. In addition, incorporating more interactive and enjoyable for students, encouraging consistent usage and progress.

Integration with Learning Management Systems (LMS) like Moodle or Canvas can make EduTutor AI part of larger institutional workflows. Furthermore, with the application of predictive analytics and machine learning, the platform could identify at-risk students and provide targeted interventions to prevent dropouts or failure. These enhancements will enable EduTutor AI to become a comprehensive, intelligent, and scalable solution for modern education challenges.

11.APPENDIX

Source Code:

requirements.txt:

fastapi

uvicorn

firebase-admin

pydantic

firebase_admin

typing

requests

uuid

datetime

streamlit

pyrebase

#note based on python version some modules will not work properly installed the required version and better to create virtual environment

Venv.txt :

replace the folder with your virtual environment if created

frontend/app.py :

```
import streamlit as st

import pyrebase

import requests

import virtualassistant

from datetime import datetime

import re

st.set_page_config(page_title="EduTutor AI", layout="wide")

# Firebase config

firebaseConfig = {

}

firebase = pyrebase.initialize_app(firebaseConfig)

auth = firebase.auth()

db = firebase.database()



# ---------- Session State ----------

if "is_logged_in" not in st.session_state:

    st.session_state["is_logged_in"] = False

if "user" not in st.session_state:

    st.session_state["user"] = None
```

```
if "role" not in st.session_state:
    st.session_state["role"] = ""
if "name" not in st.session_state:
    st.session_state["name"] = ""
# -------------------- UI --------------------
st.title("    EduTutor AI")


if not st.session_state["is_logged_in"]:
    menu = st.sidebar.selectbox("Menu", ["Login", "Signup"])
    email = st.sidebar.text_input("Email")
    password = st.sidebar.text_input("Password", type="password")


    # ---------- SIGNUP ----------
    if menu == "Signup":
        role = st.sidebar.selectbox("I am a", ["Student", "Teacher"])
        name = st.sidebar.text_input("Your Name")
        if st.sidebar.button("Create Account"):
            try:
                user = auth.create_user_with_email_and_password(email, password)
```

```
            uid = user['localId']


            db.child("users").child(uid).set({
                "email": email,
                "role": role,
                "name": name
            })
            st.success("✅ Account created successfully! Please log in.")
        except Exception as e:
            st.error(f"❌ Signup failed: {e}")


# ---------- LOGIN ----------
elif menu == "Login":
    if st.sidebar.button("Login"):
        try:
            user = auth.sign_in_with_email_and_password(email, password)
            uid = user["localId"]


            # Check if user details exist in DB
```

```
            user_data = db.child("users").child(uid).get().val()


            if user_data and "role" in user_data and "name" in user_data:
                st.session_state["user"] = user
                st.session_state["role"] = user_data["role"]
                st.session_state["name"] = user_data["name"]
                st.session_state["is_logged_in"] = True


                st.success(f"✅ Login successful! Logged in as {user_data['role']}.")
                st.rerun()  #     Force Streamlit to refresh UI
            else:
                st.error("❌ Login failed: Your account is not fully registered. Please sign up first.")
        except Exception as e:
            st.error(f"❌ Login failed: {e}")


# ---------- DASHBOARD ----------
else:
    # Default page setup
    if "page" not in st.session_state:
```

```
    st.session_state["page"] = "Dashboard"


# Sidebar navigation
st.sidebar.success(f"    Welcome {st.session_state['name']}!")
st.sidebar.write(f"Role: **{st.session_state['role']}**")


# Sidebar menu options
if st.sidebar.button("    Dashboard"):
    st.session_state["page"] = "Dashboard"
if st.sidebar.button("    Virtual Assistant"):
    st.session_state["page"] = "Virtual Assistant"
if st.sidebar.button("    Quiz Generator"):
    st.session_state["page"] = "Quiz Generator"
if st.sidebar.button("    Student History"):
    st.session_state["page"] = "Student History"
if st.sidebar.button("    Chat History"):
    st.session_state["page"] = "Chat History"
if st.sidebar.button("    AI Summarizer"):
    st.session_state["page"] = "AI Summarizer"
```

```
if st.sidebar.button("    AI Flashcards"):
    st.session_state["page"] ="AI Flashcards"
if st.sidebar.button("    Logout"):
    st.session_state["is_logged_in"] = False
    st.session_state["user"] = None
    st.session_state["role"] = ""
    st.session_state["name"] = ""
    st.session_state["page"] = "Dashboard"
    st.rerun()
# ---------- Main Content ----------
if st.session_state["page"] == "Dashboard":
    st.header("    Dashboard")
    if st.session_state["role"] == "Student":
        st.success("    You are logged in as a student.")
    elif st.session_state["role"] == "Teacher":
        st.success("    You are logged in as a teacher.")
elif st.session_state["page"] == "Virtual Assistant":
    st.header("    EduTutor AI - Virtual Assistant")
    st.write("Ask your study-related questions here...")
```

```
user_input = st.text_area("    Ask a question:")
if st.button("Get Answer") and user_input.strip():
    with st.spinner("Thinking..."):
        response = virtualassistant.ask_tutor(user_input)
    st.success("✅ Answer")
    st.write(response)
    #    Save chat to backend
    import requests
    def save_chat_to_backend(uid, question, answer):
        url = "http://localhost:8000/save-chat-history"
        payload = {"uid": uid, "question": question, "answer": answer}
        try:
            requests.post(url, json=payload)
        except Exception as e:
            st.warning(f"   Couldn't save chat: {e}")

    if "user" in st.session_state and st.session_state["user"]:
        uid = st.session_state["user"]["localId"]
        save_chat_to_backend(uid, user_input, response)
```

```
# Only run this block for the Quiz Generator page
    elif st.session_state["page"] == "Quiz Generator":
        st.title("    Edu Tutor AI -  Quiz Generator")
        st.markdown("Generate a 10-question quiz based on your topic and difficulty level.")
        # Inputs
        uid = st.session_state["user"]["localId"]
        topic = st.text_input("Enter quiz topic", "Recursion")
        difficulty = st.selectbox("Select difficulty", ["easy", "medium", "hard"])
        # Session state
        if "quiz" not in st.session_state:
            st.session_state.quiz = []
        if "submitted" not in st.session_state:
            st.session_state.submitted = False
        if "user_answers" not in st.session_state:
            st.session_state.user_answers = {}
        # Generate Quiz
        if st.button("    Generate Quiz"):
            if not uid or not topic:
                st.warning("Please provide both UID and Topic.")
```

```
else:
    with st.spinner("Generating quiz... please wait"):
        try:
            response = requests.post(
                "http://localhost:8000/generate-quiz",
                json={"uid": uid, "topic": topic, "difficulty": difficulty}
            )
            if response.status_code == 200:
                quiz = response.json()
                if len(quiz) == 10:
                    st.session_state.quiz = quiz
                    st.session_state.user_answers = {}
                    st.session_state.submitted = False
                    st.success("✅ Quiz generated successfully!")
                else:
                    st.error("Quiz must contain exactly 10 questions. Try again.")
            else:
                st.error(f"Failed to generate quiz: {response.json()['detail']}")
        except Exception as e:
```

```
            st.error(f"Error: {str(e)}")

# Quiz UI
if st.session_state.quiz and not st.session_state.submitted:
    st.subheader("    Attempt the Quiz")
    for i, q in enumerate(st.session_state.quiz):
        options_display = [f"{key}. {val}" for key, val in q["options"].items()]
        selected = st.radio(
            label=f"**Q{i+1}. {q['question']}**",
            options=options_display,
            key=f"user_q{i}"
        )
        if selected:
            st.session_state.user_answers[i] = selected[0]

    if st.button("✅ Submit Quiz"):
        if len(st.session_state.user_answers) < 10:
            st.warning("Please answer all questions before submitting.")
        else:
            st.session_state.submitted = True
```

```
# Calculate result and prepare data
correct = 0
quiz_result = []
for i, q in enumerate(st.session_state.quiz):
    user_ans = st.session_state.user_answers.get(i)
    correct_ans = q["answer"]
    if user_ans == correct_ans:
        correct += 1
    quiz_result.append({
        "question": q["question"],
        "options": q["options"],
        "answer": correct_ans,
        "user_answer": user_ans
    })
timestamp = datetime.now().isoformat()
history_payload = {
    "uid": uid,
    "topic": topic,
    "difficulty": difficulty,
```

```
                "score": correct,
                "total": 10,
                "timestamp": timestamp,
                "quiz": quiz_result
            }
            try:
                save_resp = requests.post(
                    "http://localhost:8000/save-quiz-history",
                    json=history_payload
                )
                if save_resp.status_code == 200:
                    st.success("    Quiz history saved successfully!")
                else:
                    st.error("❌ Failed to save quiz history.")
            except Exception as e:
                st.error(f"Error saving quiz history: {str(e)}")
# Show results
if st.session_state.quiz and st.session_state.submitted:
    st.subheader("    Quiz Results")
```

```
correct = 0
for i, q in enumerate(st.session_state.quiz):
    correct_ans = q["answer"]
    selected = st.session_state.user_answers.get(i, "")
    is_correct = (selected == correct_ans)
    st.markdown(f"**Q{i+1}. {q['question']}**")
    for opt_key, opt_val in q["options"].items():
        label = f"{opt_key}. {opt_val}"
        if opt_key == correct_ans and opt_key == selected:
            st.success(f"✓ {label} (Correct Answer & You Selected)")
        elif opt_key == correct_ans:
            st.info(f"   {label} (Correct Answer)")
        elif opt_key == selected:
            st.error(f"✗ {label} (Your Answer)")
        else:
            st.write(label)
    st.markdown("---")
    if is_correct:
        correct += 1
```

```
        st.info(f"    You scored **{correct} out of 10**.")


elif st.session_state["page"] == "Student History" :
    st.header("     student Quiz History")
    st.write("Here is your performance summary.")
    import streamlit as st
    import requests
    from datetime import datetime
    st.title("    Your Quiz History")
    uid = st.session_state["user"]["localId"]
    try:
        with st.spinner("Fetching quiz history..."):
            res = requests.get(f"http://localhost:8000/student-history?uid={uid}")
            if res.status_code == 200:
                history = res.json()
                if not history:
                    st.info("No quiz history found. Try taking a quiz first!")
                else:
                    for idx, record in enumerate(history[::-1]):
```

```
                with st.expander(f"    {record['timestamp']} | {record['topic']} | {record['difficulty']} | Score: {record['score']}/10"):
                    for i, q in enumerate(record["quiz"]):
                        st.markdown(f"**Q{i+1}. {q['question']}**")
                        for opt_key, opt_val in q["options"].items():
                            label = f"{opt_key}. {opt_val}"
                            if opt_key == q["answer"] and opt_key == q["user_answer"]:
                                st.success(f"✅ {label} (Correct Answer & You Selected)")
                            elif opt_key == q["answer"]:
                                st.info(f"   {label} (Correct Answer)")
                            elif opt_key == q["user_answer"]:
                                st.error(f"❌ {label} (Your Answer)")
                            else:
                                st.write(label)
                        st.markdown("---")
            else:
                st.error("❌ Failed to load quiz history.")
    except Exception as e:
        st.error(f"Error: {str(e)}")
elif st.session_state["page"] == "Chat History":
```

```
st.header("    Your Chat History")
with st.spinner("Loading Previous chat... please wait"):
    try:
        import requests
        uid = st.session_state["user"]["localId"]
        res = requests.get(f"http://localhost:8000/get-chat-history/{uid}")
        if res.status_code == 200:
                response = requests.get(f"http://localhost:8000/get-chat-history/{uid}")
                if response.status_code == 200:
                    data = response.json()
                    if data:
                        for i, chat in enumerate(data.values()):
                            st.write(f"**Q{i+1}:** {chat['question']}")
                            st.write(f"**A{i+1}:** {chat['answer']}")
                    else:
                        st.info("No chat history found.")

                else:
                            st.info("No chat history found.")
```

```
            else:
                st.error("Failed to load chat history.")
        except Exception as e:
            st.error(f"   Error fetching chat history: {e}")
elif st.session_state["page"] == "AI Summarizer":
    API_BASE = "http://localhost:8000"  # Change if deployed
    st.header("    EduTutor AI - AI Summarizer")
    st.write("Paste your notes below and get a concise 5-point summary!")
    user_notes = st.text_area("   Enter your study notes here")
    if st.button("Summarize"):
        if not user_notes.strip():
            st.warning("Please enter some text to summarize.")
        else:
            with st.spinner("Generating summary..."):
                try:
                    response = requests.post(
                        f"http://localhost:8000/summarize-notes",
                        json={"notes": user_notes}
                    )
```

```
if response.status_code == 200:
    raw_summary = response.json()["summary"]
    #     Clean and standardize the summary
    import re
    raw_lines = re.findall(r'\d+\.\s+.*?(?=\n\d+\.|\Z)', raw_summary.strip(), re.DOTALL)
    seen = set()
    cleaned = []
    for line in raw_lines:
        sentence = re.sub(r'\s+', ' ', line.strip())  # normalize whitespace
        if sentence not in seen and len(sentence.split()) > 3:
            cleaned.append(sentence)
            seen.add(sentence)
    # ✅ Display final cleaned summary (max 7 points)
    st.success("Here’s your summary:")
    for i, point in enumerate(cleaned[:7], start=1):
        # Remove previous numbering and reformat
        content = re.sub(r'^\d+\.\s*', '', point)
        st.markdown(f"✅    **{i}. {content}**")
else:
```

```
                    st.error(f"Error: {response.text}")
            except Exception as e:
                st.error(f"Request failed: {e}")
elif st.session_state["page"] == "AI Flashcards":
    st.header("    EduTutor AI - Flashcard Generator")
    st.write("Turn your study notes into interactive flashcards!")
    API_BASE = "http://localhost:8000"
    user_notes = st.text_area("    Paste your study notes here")
    if st.button("Generate Flashcards"):
        if not user_notes.strip():
            st.warning("Please enter some notes.")
        else:
            with st.spinner("Generating flashcards..."):
                try:
                    response = requests.post(
                        f"{API_BASE}/generate-flashcards",
                        json={"notes": user_notes}
                    )
                    if response.status_code == 200:
```

```
                    flashcards_raw = response.json()["flashcards"]

                    # Extract Q&A pairs

                    pairs = re.findall(r"Q:\s*(.*?)\nA:\s*(.*?)(?=\nQ:|\Z)", flashcards_raw, re.DOTALL)

                    st.success("Here are your flashcards:")

                    for i, (q, a) in enumerate(pairs, 1):

                        with st.expander(f"Flashcard {i}: {q.strip()}"):

                            st.write(f"**Answer:** {a.strip()}")

                else:

                    st.error(f"Error: {response.text}")

            except Exception as e:

                st.error(f"Request failed: {e}")
```

frontend/virtualassistant.py :

```
import os

import re

import requests

from dotenv import load_dotenv


load_dotenv()
```

```
API_KEY = os.getenv("API_KEY")
PROJECT_ID = os.getenv("PROJECT_ID")
REGION = os.getenv("REGION")
MODEL_ID = os.getenv("MODEL_ID")

import requests
def get_iam_token():
    url = "https://iam.cloud.ibm.com/identity/token"
    data = {
        "apikey": "Replace with Your IBM key",
        "grant_type": "urn:ibm:params:oauth:grant-type:apikey"
    }
    headers = {
        "Content-Type": "application/x-www-form-urlencoded"
    }

    response = requests.post(url, data=data, headers=headers)
    response.raise_for_status()  # This will raise HTTPError if status != 200

    return response.json()["access_token"]
def ask_tutor(user_prompt):
    token = get_iam_token()
    url=f"https://{REGION}.ml.cloud.ibm.com/ml/v1/text/generation?version=2024-05-29"
```

```
headers = {
    "Authorization": f"Bearer {token}",
    "Content-Type": "application/json"
}

# System instruction to restrict responses
system_instruction = (
    "You are EduTutor AI, an educational assistant. "
    "Only answer questions related to academic subjects, study help, exams, or learning. "
    "If the user asks something unrelated to education, politely say you're restricted to education-related topics."
)

# Construct full prompt
full_prompt = (
    f"{system_instruction}\n\n"
    f"User: {user_prompt}\n"
    f"AI:"
)

payload = {
    "model_id": MODEL_ID,
    "input": full_prompt,
    "project_id": PROJECT_ID,
    "parameters": {
```

```
        "decoding_method": "sample",
        "temperature": 0.7,
        "top_p": 0.9,
        "max_new_tokens": 3000,
        "stop_sequences": ["User:", "AI:"]
    }
}

response = requests.post(url, headers=headers, json=payload)
result = response.json()

if "results" in result:
        output = result["results"][0]["generated_text"].strip()
        # Remove anything after 'User:' or 'AI:' if it still appears
        output = re.split(r"\bUser:|\bAI:", output)[0].strip()
        return output

elif "errors" in result:
    return f"❌ API Error: {result['errors'][0]['message']}"
else:
    return f"⚠️ Unknown error: {result}"
```

backend/main.py :

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Dict
import requests
import re
from datetime import datetime
import firebase_admin
from firebase_admin import credentials, db
from fastapi.middleware.cors import CORSMiddleware
import uuid
# Firebase init (use your JSON credentials)
#create a service account in firebase
cred = credentials.Certificate("?Your serviceAccountKey.json")
firebase_admin.initialize_app(cred, {
    'databaseURL': 'replace with your database URL'
})

class ChatEntry(BaseModel):
    uid: str
    question: str
    answer: str
# FastAPI app
app = FastAPI()
```

```
# Request model
class QuizRequest(BaseModel):
    uid: str
    topic: str
    difficulty: str = "medium"

# Response model
class QuizQuestion(BaseModel):
    question: str
    options: Dict[str, str]
    answer: str

# Allow frontend access (adjust origins in production)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class Question(BaseModel):
    question: str
```

```
    options: Dict[str, str]
    answer: str
    user_answer: str


class QuizHistoryRequest(BaseModel):
    uid: str
    score: int
    total: int
    topic: str
    difficulty: str
    timestamp: str
    quiz: List[Question]  # List of 10 questions with user answers
class NotesRequest(BaseModel):
    notes: str
class FlashcardRequest(BaseModel):
    notes: str


# IBM Granite API configuration
API_KEY = "Replace With your IBM API KEY"
MODEL_ID = "ibm/granite-3-2b-instruct"
PROJECT_ID = "Project ID IN IBM CLoud"
VERSION = "2024-05-01"
#replace this end point URl according to your model
ENDPOINT = f"https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version={VERSION}"
```

```
# Get IAM token
def get_iam_token(api_key):
    response = requests.post(
        "https://iam.cloud.ibm.com/identity/token",
        data={"apikey": api_key, "grant_type": "urn:ibm:params:oauth:grant-type:apikey"},
        headers={"Content-Type": "application/x-www-form-urlencoded"}
    )
    if response.status_code == 200:
        return response.json()["access_token"]
    else:
        raise Exception(f"IAM Token error: {response.text}")

# Call IBM model and generate quiz text
def generate_quiz_text(topic, difficulty):
    token = get_iam_token(API_KEY)
    prompt = f"""
    Generate exactly 10 multiple-choice questions on the topic: "{topic}" with {difficulty} difficulty.
    Each question must have:

    - Four options (A, B, C, D) on separate lines.
    - A clearly labeled correct answer in this format: Answer: <A/B/C/D>

    Do NOT include any code formatting, Markdown, backticks, or special characters. Strictly use plain text like this:
```

```
Q1. <Question>
A. Option A
B. Option B
C. Option C
D. Option D
Answer: <Correct Option Letter>

Repeat for 10 questions only.
"""



headers = {
    "Authorization": f"Bearer {token}",
    "Content-Type": "application/json"
}

data = {
    "model_id": MODEL_ID,
    "input": prompt.strip(),
    "parameters": {
        "max_new_tokens": 2000,
        "temperature": 0.7
    },
```

```
        "project_id": PROJECT_ID
    }

    response = requests.post(ENDPOINT, headers=headers, json=data)
    result = response.json()

    if "results" in result:
        return result["results"][0]["generated_text"]
    else:
        raise ValueError(f"Unexpected response: {result}")

# Parse model output into structured questions
def parse_quiz_output(quiz_text: str) -> List[QuizQuestion]:
    questions = []
    # Split using Q1., Q2., ..., but also support without exact dots
    blocks = re.split(r"Q\d+\.\s*", quiz_text.strip())
    for block in blocks:
        if not block.strip():
            continue

        lines = block.strip().split("\n")
        if len(lines) < 6:
            continue  # Incomplete block
```

```
    question_line = lines[0].strip()
    try:
        options = {}
        for i, opt in enumerate(["A", "B", "C", "D"]):
            if not lines[i + 1].strip().startswith(f"{opt}."):
                raise ValueError("Missing or malformed option")
            options[opt] = lines[i + 1].strip()[2:].strip()

        # Find Answer:
        answer_line = next((line for line in lines if "Answer:" in line), "")
        answer_match = re.search(r"Answer:\s*([A-D])", answer_line)
        if not answer_match:
            raise ValueError("No valid answer found")
        answer = answer_match.group(1)

        questions.append(QuizQuestion(
            question=question_line,
            options=options,
            answer=answer
        ))

    except Exception:
        continue  # Skip malformed question
```

```
    return questions


# API endpoint for quiz generation
@app.post("/generate-quiz", response_model=List[QuizQuestion])
def generate_quiz(request: QuizRequest):
    try:
        attempts = 0
        max_attempts = 3
        structured_quiz = []

        while attempts < max_attempts:
            raw_quiz = generate_quiz_text(request.topic, request.difficulty)
            structured_quiz = parse_quiz_output(raw_quiz)

            if len(structured_quiz) == 10:
                break  # Success
            attempts += 1

        if len(structured_quiz) != 10:
            raise HTTPException(status_code=500, detail=f"Failed to generate 10 questions after {max_attempts} attempts.")

        return structured_quiz

    except Exception as e:
```

```
        raise HTTPException(status_code=500, detail=str(e))



@app.post("/save-chat-history")
def save_chat(entry: ChatEntry):
    try:
        timestamp = datetime.utcnow().isoformat()
        db.reference(f'chat_history/{entry.uid}').push({
            'question': entry.question,
            'answer': entry.answer,
            'timestamp': timestamp
        })
        return {"status": "success"}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
@app.get("/get-chat-history/{uid}")
def get_history(uid: str):
    try:
        ref = db.reference(f'chat_history/{uid}')
        return ref.get() or {}
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
@app.get("/student-history")
def get_student_history(uid: str):
```

```
    ref = db.reference(f"quiz_history/{uid}")
    records = ref.get()
    if not records:
        return []
    return list(records.values())



# ================================
# Save Quiz History Endpoint
# ================================
@app.post("/save-quiz-history")
def save_quiz_history(request: QuizHistoryRequest):
    try:
        ref = db.reference(f"quiz_history/{request.uid}")
        history_id = str(uuid.uuid4())

        ref.child(history_id).set({
            "score": request.score,
            "total": request.total,
            "topic": request.topic,
            "difficulty": request.difficulty,
            "timestamp": request.timestamp,
            "quiz": [q.dict() for q in request.quiz]
        })
```

```
        return {"message": "✅ Quiz history saved successfully."}
    except Exception as e:
        return {"error": str(e)}


# ================================
# Get Quiz History by UID
# ================================
@app.get("/get-quiz-history/{uid}")
def get_quiz_history(uid: str):
    try:
        ref = db.reference(f"quiz_history/{uid}")
        data = ref.get()
        if data:
            return data
        return {"message": "No quiz history found."}
    except Exception as e:
        return {"error": str(e)}


def clean_summary(text: str) -> str:
    # Remove bullets like ✅, dashes, etc.
    text = re.sub(r'✅|\s*-\s*', '', text)


    # Extract lines that look like numbered points
```

```
    lines = re.findall(r'(?:^\d{1,2}\.\s.*?$)', text, flags=re.MULTILINE)

    # Re-number them properly
    cleaned = ""
    for idx, line in enumerate(lines, start=1):
        content = re.sub(r'^\d{1,2}\.\s*', '', line)
        cleaned += f"{idx}. {content.strip()}\n"

    return cleaned.strip()

@app.post("/summarize-notes")
def summarize_notes(req: NotesRequest):
    token = get_iam_token(API_KEY)

    # Improved prompt to guide clean output
    prompt = (
        "Summarize the following text into exactly 5 to 7 clear and concise points. "
        "Each point should be numbered like 1., 2., 3. Do not include emojis or bullets.\n\n"
        f"{req.notes}"
    )

    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {token}",
```

```
}

data = {
    "model_id": MODEL_ID,
    "input": prompt.strip(),
    "parameters": {
        "max_new_tokens": 300,
        "temperature": 0.5
    },
    "project_id": PROJECT_ID
}

response = requests.post(ENDPOINT, headers=headers, json=data)

if response.status_code != 200:
    raise HTTPException(status_code=response.status_code, detail=response.text)

try:
    summary_raw = response.json()["results"][0]["generated_text"]
    summary_cleaned = clean_summary(summary_raw)
    return {"summary": summary_cleaned}
except Exception:
    raise HTTPException(status_code=500, detail="Unexpected response format from IBM Granite.")
```

```
@app.post("/generate-flashcards")
def generate_flashcards(req: FlashcardRequest):
    token = get_iam_token(API_KEY)

    prompt = (
        "From the following study notes, generate 5 question-answer flashcards. "
        "Each flashcard should be numbered and follow this format:\n\n"
        "Q: <Question>\nA: <Answer>\n\n"
        f"{req.notes}"
    )

    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {token}",
    }

    data = {
        "model_id": MODEL_ID,
        "input": prompt.strip(),
        "parameters": {
            "max_new_tokens": 400,
            "temperature": 0.5
        },
        "project_id": PROJECT_ID
```

```
    }

    response = requests.post(ENDPOINT, headers=headers, json=data)

    if response.status_code != 200:
        raise HTTPException(status_code=response.status_code, detail=response.text)

    try:
        result = response.json()["results"][0]["generated_text"]
        return {"flashcards": result.strip()}
    except Exception:
        raise HTTPException(status_code=500, detail="Unexpected flashcard format.")
```

Data Base link:

Github link:

https://github.com/Sowmyabharathimullapudi/EduTutor-AI