Project Documentation – Personalized Learning Tutor AI

1. Introduction

• Project Title: Personalized Learning Tutor AI

• Team Members:

- [A.Sivasubramanian]

- [T.Sriram]

- [R.Kathiravan]

-[K.Karthik]

2. Project Overview

Purpose:
The Personalized Learning Tutor AI is designed to provide students with a customized and adaptive learning experience. Using AI and analytics, it adjusts study material, practice questions, and feedback based on each learner's pace, strengths, and weaknesses. It acts as a virtual tutor—offering guidance, doubt clarification, progress tracking, and motivational support.

Features:
- Conversational Interface: Natural language chat for doubt-solving and learning guidance.
- Adaptive Learning Paths: Automatically adjusts difficulty level and recommends next topics based on learner performance.
- Content Summarization: Summarizes large textbooks or PDFs into key notes for quick understanding.
- Personalized Quizzes & Assessments: Generates practice questions and evaluates progress with instant feedback.
- Study Recommendations: Suggests resources (videos, notes, examples) based on learner's weak areas.
- Progress Dashboard: Tracks learner's performance, completion rate, and skill improvement.

- Feedback & Motivation: Provides learning tips and motivational messages to keep students engaged.
- Multimodal Input Support: Accepts text, PDFs, or images (handwritten notes) for learning support.

3. Architecture

- Frontend (Streamlit/Gradio): Interactive UI for students with dashboards, quizzes, progress trackers, and chat.
- Backend (FastAPI): Handles quiz generation, adaptive learning algorithms, document summarization, and chat responses.
- LLM Integration (IBM Watsonx / OpenAI / Similar): Used for natural language understanding, summarization, and tutor-style responses.
- Vector Search (Pinecone / FAISS): Embeds study material for semantic search and instant doubt-solving.
- ML Modules (Adaptive Engine): Learner profiling and recommendation engine for personalized study paths.

4. Setup Instructions

Prerequisites:
- Python 3.9+
- pip and venv tools
- API keys (LLM provider, vector database)
- Internet connection

Installation Steps:
1. Clone repository
2. Install dependencies from requirements.txt
3. Create .env file with API keys
4. Run FastAPI backend
5. Launch Streamlit frontend
6. Upload study materials and interact with tutor

5. Folder Structure

app/ – Backend logic (FastAPI routes, models, algorithms)
app/api/ – Modular API endpoints (chat, quiz, summary, feedback)
ui/ – Frontend components (Streamlit/Gradio pages, dashboards)
adaptive_engine.py – Learner profiling & recommendation engine
quiz_generator.py – AI-based question generation
doc_summarizer.py – Summarizes textbooks & PDFs

progress_tracker.py – Tracks learning metrics
report_generator.py – Generates personalized reports

6. Running the Application

1. Start FastAPI backend
2. Launch Streamlit dashboard
3. Navigate using sidebar (Chat, Quizzes, Dashboard, Resources)
4. Upload textbooks or notes
5. Interact with tutor for Q&A, summaries, or practice

7. API Documentation

- POST /chat/ask – Learner asks question, AI responds
- POST /upload-doc – Uploads notes/books for summarization & search
- GET /recommend-topics – Suggests topics based on learner progress
- GET /generate-quiz – Creates quiz based on subject & difficulty
- POST /submit-feedback – Stores learner feedback

8. Authentication

- Token-based authentication (JWT or API keys)
- Role-based access (Student, Teacher, Admin)
- Planned: Session history and personalized learning logs

9. User Interface

- Sidebar navigation
- Progress charts & skill graphs
- Tabs for Chat, Quiz, Notes, Recommendations
- Real-time interaction
- Option to download study reports

10. Testing

- Unit Testing: Adaptive engine, quiz generation
- API Testing: Swagger UI, Postman
- Manual Testing: Quizzes, uploads, chat flow
- Edge Cases: Incorrect answers, missing data, large textbook inputs

11. Future Enhancements

- Voice-based tutoring
- Multilingual learning support

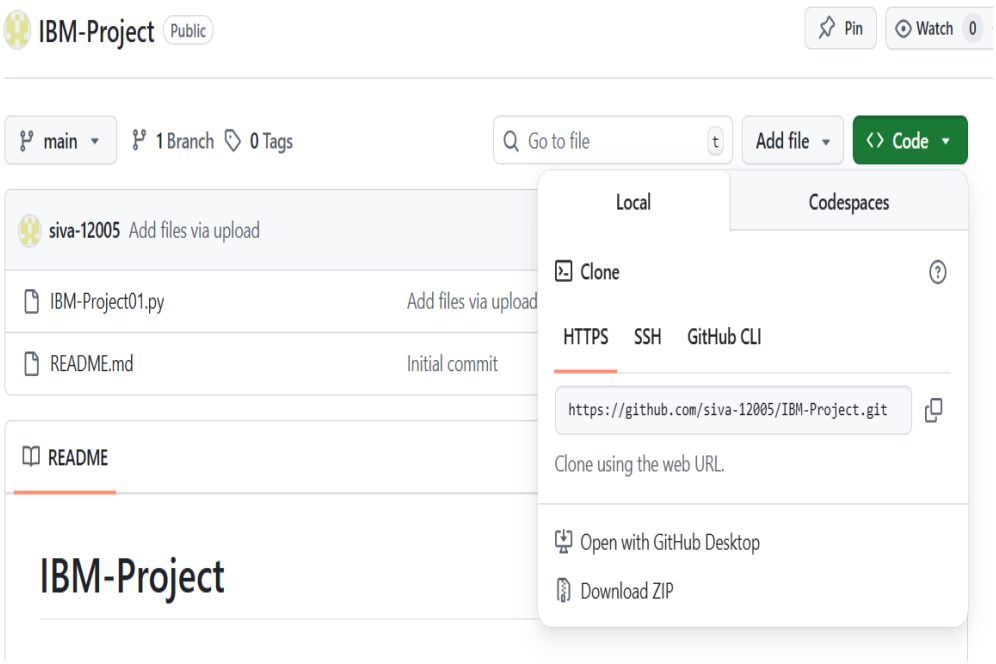
- Gamified learning experience (badges, rewards)
- Integration with Learning Management Systems (LMS)

PROJECT SCREENSHOT

```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=512):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
```

SCREENSHOT 2

SCREENSHOT 3



Thank You