import json

import re

from typing import List, Dict, Tuple

from dataclasses import dataclass

import math

@dataclass

class StudyMaterial:

title: str

content: str

topic: str

class SimpleStudyCompanion:

def \_\_init\_\_(self):

self.syllabus = []

self.textbook\_content = []

self.study\_progress = {}

def add\_syllabus\_item(self, title: str, content: str, topic: str):

"""Add a syllabus item to the knowledge base"""

self.syllabus.append(StudyMaterial(title, content, topic))

def add\_textbook\_content(self, title: str, content: str, topic: str):

"""Add textbook content to the knowledge base"""

self.textbook\_content.append(StudyMaterial(title, content, topic))

def keyword\_search(self, query: str, materials: List[StudyMaterial]) -> List[StudyMaterial]:

"""Simple keyword-based search"""

query\_words = query.lower().split()

relevant\_materials = []

for material in materials:

content\_lower = material.content.lower()

title\_lower = material.title.lower()

# Check if any query words appear in title or content

if any(word in content\_lower or word in title\_lower for word in query\_words):

relevant\_materials.append(material)

return relevant\_materials

def semantic\_search(self, query: str) -> List[StudyMaterial]:

"""Simple semantic search using topic matching"""

query\_lower = query.lower()

relevant\_materials = []

# Simple topic matching

for material in self.syllabus + self.textbook\_content:

if material.topic.lower() in query\_lower or query\_lower in material.topic.lower():

relevant\_materials.append(material)

return relevant\_materials

def hybrid\_search(self, query: str) -> List[StudyMaterial]:

"""Combine keyword and semantic search"""

keyword\_results = self.keyword\_search(query, self.syllabus + self.textbook\_content)

semantic\_results = self.semantic\_search(query)

# Combine and deduplicate

all\_results = keyword\_results + semantic\_results

unique\_results = []

seen\_titles = set()

for result in all\_results:

if result.title not in seen\_titles:

unique\_results.append(result)

seen\_titles.add(result.title)

return unique\_results[:3] # Return top 3 results

def is\_math\_query(self, query: str) -> bool:

"""Check if query contains mathematical operations"""

math\_keywords = ['calculate', 'solve', 'equation', 'formula', '+', '-', '\*', '/', '=']

return any(keyword in query.lower() for keyword in math\_keywords)

def simple\_calculator(self, expression: str) -> str:

"""Basic calculator for simple mathematical expressions"""

try:

# Extract mathematical expression from query

math\_pattern = r'[\d\+\-\\*/\(\)\.\s]+'

matches = re.findall(math\_pattern, expression)

for match in matches:

if len(match.strip()) > 3: # Avoid single digits

# Safety check - only allow basic operations

safe\_chars = set('0123456789+-\*/.() ')

if all(c in safe\_chars for c in match):

result = eval(match.strip())

return f"Calculation result: {result}"

return "Could not parse mathematical expression"

except:

return "Error in calculation"

def answer\_question(self, question: str) -> str:

"""Main function to answer student questions"""

# Check if it's a math question

if self.is\_math\_query(question):

calc\_result = self.simple\_calculator(question)

math\_context = self.hybrid\_search(question)

response = f"{calc\_result}\n\n"

if math\_context:

response += "Related course material:\n"

for material in math\_context:

response += f"- {material.title}: {material.content[:100]}...\n"

return response

# Regular question answering

relevant\_materials = self.hybrid\_search(question)

if not relevant\_materials:

return "I couldn't find relevant information in your course materials. Could you rephrase your question?"

response = "Based on your course materials:\n\n"

for material in relevant\_materials:

response += f"From '{material.title}' ({material.topic}):\n"

response += f"{material.content}\n\n"

return response

def generate\_quiz(self, topic: str, num\_questions: int = 3) -> List[Dict]:

"""Generate simple quiz questions based on topic"""

relevant\_materials = [m for m in self.syllabus + self.textbook\_content

if topic.lower() in m.topic.lower()]

quiz\_questions = []

for i, material in enumerate(relevant\_materials[:num\_questions]):

# Simple question generation based on content

question = {

'question': f"What is the main concept in '{material.title}'?",

'topic': material.topic,

'reference': material.content[:200] + "..." if len(material.content) > 200 else material.content

}

quiz\_questions.append(question)

return quiz\_questions

def get\_study\_hint(self, topic: str) -> str:

"""Provide study hints for a given topic"""

relevant\_materials = [m for m in self.syllabus + self.textbook\_content

if topic.lower() in m.topic.lower()]

if not relevant\_materials:

return f"No materials found for topic: {topic}"

hint = f"Study hint for {topic}:\n\n"

hint += f"Key concept: {relevant\_materials[0].content[:150]}...\n\n"

hint += "Try to understand the main ideas and practice with examples."

return hint

def suggest\_next\_topic(self, current\_topic: str) -> str:

"""Suggest next topic to study"""

topics = list(set([m.topic for m in self.syllabus]))

if current\_topic in topics:

current\_index = topics.index(current\_topic)

if current\_index < len(topics) - 1:

return f"Next suggested topic: {topics[current\_index + 1]}"

return "Consider reviewing all topics or moving to practice questions."

# Example usage and demonstration

def demo\_study\_companion():

"""Demonstrate the study companion functionality"""

companion = SimpleStudyCompanion()

# Add sample course materials

companion.add\_syllabus\_item(

"Introduction to Algebra",

"Algebra is the study of mathematical symbols and the rules for manipulating these symbols. It includes solving equations and working with variables.",

"Mathematics"

)

companion.add\_syllabus\_item(

"Geometry Basics",

"Geometry deals with shapes, sizes, and properties of space. Key concepts include area, perimeter, and volume calculations.",

"Mathematics"

)

companion.add\_textbook\_content(

"Quadratic Equations",

"A quadratic equation is in the form ax² + bx + c = 0. Solutions can be found using the quadratic formula: x = (-b ± √(b²-4ac)) / 2a",

"Mathematics"

)

companion.add\_textbook\_content(

"Linear Functions",

"Linear functions have the form y = mx + b, where m is the slope and b is the y-intercept. They graph as straight lines.",

"Mathematics"

)

print("=== AI Study Companion Demo ===\n")

# Test question answering

print("1. Question Answering:")

question = "What is a quadratic equation?"

print(f"Q: {question}")

print(f"A: {companion.answer\_question(question)}")

# Test math calculation

print("\n2. Math Problem Solving:")

math\_question = "Calculate 15 + 25 \* 2"

print(f"Q: {math\_question}")

print(f"A: {companion.answer\_question(math\_question)}")

# Test quiz generation

print("\n3. Quiz Generation:")

quiz = companion.generate\_quiz("Mathematics", 2)

for i, q in enumerate(quiz, 1):

print(f"Question {i}: {q['question']}")

print(f"Reference: {q['reference']}\n")

# Test study hints

print("4. Study Hint:")

hint = companion.get\_study\_hint("Mathematics")

print(hint)

# Test topic suggestions

print(f"\n5. Next Topic Suggestion:")

suggestion = companion.suggest\_next\_topic("Mathematics")

print(suggestion)

if \_\_name\_\_ == "\_\_main\_\_":

demo\_study\_companion()