**Name: Md Tareq Shah Alam**

**Role Applied: Data Champion**

**Assignment Chosen: AI Agent Development**

**Timeline: I will complete the assignment by 18 September 2025, 05:47 PM**

# AI Agent Development Assignment – Approach Document

## SECTION 1: BASIC DETAILS

Name: Md Tareq Shah Alam

AI Agent Title / Use Case: AI Agent to Help Students Revise for Exams

## SECTION 2: PROBLEM FRAMING

2.1. What problem does your AI Agent solve?  
Many students struggle to organize their revision, especially when they don’t know which topics to prioritize or how to test themselves effectively. This agent solves that by generating customized revision plans and quizzes.

2.2. Why is this agent useful?  
It gives students structure and reduces stress by guiding them on what to study each day, providing active recall quizzes, and tracking which topics they’ve mastered or need to review again.

2.3. Who is the target user?  
Undergraduate and high-school students preparing for upcoming exams.

2.4. What not to include?  
- No full course lectures or video lessons (keeps it light)  
- No grade predictions  
- No plagiarism-heavy answers to direct exam questions

## SECTION 3: 4-LAYER PROMPT DESIGN

### 3.1 INPUT UNDERSTANDING

Prompt:  
"You are a helpful exam revision assistant. The user will tell you their subject and goal (e.g., ‘I’m revising biology for my finals’ or ‘I want to focus on calculus integration’). Read their input and extract:  
- Subject  
- Specific topics  
- Exam timeline  
- Preferred study style (if mentioned)  
Return this as structured JSON."

Responsible for: Extract key details from vague user requests into structured data for planning.

Example Input: "I have a math exam in 2 weeks. Need help revising algebra and calculus."  
Example Output:  
{"subject":"math","topics":["algebra","calculus"],"timeline":"2 weeks","style":"not specified"}

### 3.2 STATE TRACKER

Prompt:  
"Remember the student’s subject, topics, and timeline from previous turns. Keep a running list of what they’ve already revised and what’s pending. Store this as a progress log (date, topic, status: pending/complete). Always update this log silently before responding."

How it helps the agent remember: Simulates memory of past interactions (like which topics are done) so it can personalize plans.  
Simulated by using a persistent system message with a variable `progress\_log` to store progress.

### 3.3 TASK PLANNER

Prompt:  
"Using the extracted subject, topics, and exam timeline, break down the revision into daily plans.  
Steps:  
1. Calculate how many study days remain.  
2. Distribute topics across days.  
3. Insert short daily quizzes after each topic.  
4. Update progress\_log as they complete each part.  
Return the plan in a clear daily schedule format."

Responsible for: This prompt creates the logic and structure for the study plan, acting like the “brain” of the agent.  
Planning Style: Linear chaining (Topic → Plan → Quizzes).

### 3.4 OUTPUT GENERATOR

Prompt:  
"Present the study plan in a clear checklist table with: Day, Topic, and Quiz Link. Use a friendly and motivating tone. At the end, encourage the student briefly."

Behavior & Formatting:  
- Uses Markdown tables for readability  
- Adds emojis to make it feel human and fun  
- Ends with a motivational note

## SECTION 4: CHATGPT EXPLORATION LOG

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Attempt | Prompt Variant | What Happened | What You Changed | Why You Changed It |
| 1 | Make a study plan for math | Output too generic | Added subject + timeline extraction step | To structure data for planning |
| 2 | Track completed topics | It forgot past turns | Created progress\_log variable in system prompt | To simulate persistent memory |
| 3 | Create plan + quizzes | Plan was too long | Broke it into daily slots | To make plan manageable and student-friendly |

## SECTION 5: OUTPUT TESTS

Test 1 (Normal): Input: "I have a biology exam in 10 days. Focus on genetics and cell biology."  
Output: Day 1: Genetics basics → Quiz ... Day 2: Genetics practice → Quiz ... etc.

Test 2 (Vague): Input: "Help me revise"  
Output: "Sure! Please tell me your subject, topics, and when your exam is so I can create a perfect plan."

Test 3 (Empty): Input: ""  
Output: "Looks like you didn’t mention your subject or exam date. Could you share that so I can help?"

## SECTION 6: REFLECTION

6.1 Hardest part: Getting the agent to “remember” previous turns without real memory. Solved it by simulating memory with a variable-based log.

6.2 Most enjoyable: Designing the output format — making it colorful and encouraging so students feel supported.

6.3 What to improve: Add adaptive difficulty — as the student scores well on quizzes, the plan could auto-level up.

6.4 What I learned: How powerful prompt design can be. Even without coding, structured thinking turns ChatGPT into a usable agent.

6.5 Feeling stuck: Yes, especially during the state tracker design. I broke it into smaller steps, researched how others simulate memory, and tested mini-prototypes.

## SECTION 7: HACK VALUE

- Added a progress\_log simulation so the agent behaves like it remembers user progress.  
- Created small daily quizzes as part of the output, which makes it more interactive and not just static advice.

## SECTION 5.1: ADVANCED CODE SAMPLE

# Advanced AI Exam Revision Agent (Prototype)  
# Requires: openai (optional), sklearn (optional)  
import os  
import json  
from datetime import date, timedelta  
  
OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY", "") # set this in your environment to enable LLM  
use\_llm = bool(OPENAI\_API\_KEY)  
  
def parse\_input(user\_text):  
 parts = {"subject":"","topics":[], "days":0, "style":""}  
 tokens = [t.strip() for t in user\_text.split(';') if t.strip()]  
 for t in tokens:  
 if "days" in t or "week" in t or "day" in t:  
 nums = [int(s) for s in t.split() if s.isdigit()]  
 if nums:  
 parts["days"] = nums[0]  
 elif "topic" in t or "topics" in t or ',' in t:  
 ts = t.replace("topics:","").replace("topic:","").strip()  
 parts["topics"] = [x.strip() for x in ts.split(',') if x.strip()]  
 elif "style" in t:  
 parts["style"] = t.split(':',1)[1].strip()  
 else:  
 if not parts["subject"]:  
 parts["subject"] = t  
 if parts["days"]==0:  
 parts["days"]=7  
 return parts  
  
def sm2\_schedule(reviews, days):  
 schedule = []  
 day = 0  
 for i,topic in enumerate(reviews):  
 interval = max(1, (i%days)+1)  
 schedule.append({"day": (day%days)+1, "topic": topic, "interval": interval})  
 day += interval  
 return schedule  
  
def local\_quiz\_generator(topic, kb=None):  
 if kb and topic in kb:  
 q = kb[topic][:5]  
 return [f"Q{i+1}: {qq}" for i,qq in enumerate(q)]  
 return [f"Q1: Define {topic}.", f"Q2: Key concept of {topic}?", f"Q3: Example problem on {topic}"]  
  
def build\_plan(parsed, progress\_log=None, kb=None):  
 topics = parsed["topics"] or ["General Revision"]  
 days = parsed["days"]  
 plan = []  
 if use\_llm:  
 try:  
 import openai  
 openai.api\_key = OPENAI\_API\_KEY  
 system = "You are a concise study planner. Output JSON with 'daily' list of {day,topic,quiz\_questions}."  
 prompt = f"Subject: {parsed['subject']}\nTopics: {topics}\nDays: {days}\nStyle: {parsed['style']}\nProduce a compact daily schedule."  
 resp = openai.ChatCompletion.create(model="gpt-4o-mini", messages=[{"role":"system","content":system},{"role":"user","content":prompt}], max\_tokens=800, temperature=0.2)  
 txt = resp['choices'][0]['message']['content']  
 try:  
 out = json.loads(txt)  
 return out  
 except:  
 pass  
 except Exception as e:  
 pass  
 evenly = []  
 for i in range(days):  
 topic = topics[i % len(topics)]  
 qs = local\_quiz\_generator(topic, kb)  
 evenly.append({"day": i+1, "topic": topic, "quiz": qs})  
 spaced = sm2\_schedule(topics, days)  
 for s in spaced:  
 for e in evenly:  
 if e["day"] == s["day"]:  
 e["priority"] = s["interval"]  
 return {"daily": evenly}  
  
def pretty\_print(plan):  
 print("\n Personalized Revision Plan\n")  
 print("| Day | Topic | Sample Quiz Questions |")  
 print("|-----|-------|------------------------|")  
 for d in plan["daily"]:  
 qtxt = " || ".join(d["quiz"][:3])  
 print(f"| {d['day']} | {d['topic']} | {qtxt} |")  
 print("\nGood luck! Keep reviewing and update progress after each session.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 user\_text = input("Enter input (format: Subject; topics: A,B,C; days: 10; style: active):\n")  
 parsed = parse\_input(user\_text)  
 sample\_kb = {  
 "genetics":["What is DNA?","Describe Mendelian inheritance","Define allele","Explain mutation","Describe replication"],  
 "cell biology":["Name organelles","Function of mitochondria","Cell membrane structure","Describe mitosis","Cell transport mechanisms"]  
 }  
 plan = build\_plan(parsed, kb=sample\_kb)  
 pretty\_print(plan)