# System Design & Coding Challenge: The Adaptive Learning Orchestrator

## The Architectural Challenge: Designing a Decoupled LLM-Ready System

The central task is to design and implement a Web API for an Assessment Generation Service. A naive, monolithic approach will not scale as we add more subjects, content types, or more sophisticated AI capabilities.  
  
Therefore, you should design your service around a decoupled, multi-stage orchestration pattern. This architecture will allow us to eventually replace simulated components with actual fine-tuned models or general-purpose LLMs without a full rewrite.  
  
Your system should conceptually (and in your code's structure) be broken down into at least two distinct, swappable components:

* The Planner:

- Responsibility: To translate high-level, potentially ambiguous user intent into a concrete, machine-readable execution plan.  
- Input: A StudentProfile and AssessmentRequest.  
- Output: An AssessmentPlan (a structured object/JSON). This plan is the contract between the Planner and the Executor.  
- Analogy: This is the "Reasoning" (the 'R' in ReAct) or "Chain of Thought" step. It decides what to do and why.

* The Executor:

- Responsibility: To flawlessly execute a given AssessmentPlan. It should be "dumb" in the sense that it doesn't make strategic decisions.  
- Input: An AssessmentPlan.  
- Output: A GeneratedAssessment containing the selected problems.  
- Analogy: This is the "Acting" (the 'A' in ReAct) step. It interacts with tools (in this case, the problem database) to fulfill the plan.

## The Implementation Task

You are to build a Web API with a single, primary endpoint: POST /api/assessments/generate.  
  
API Input (POST /api/assessments/generate):  
A JSON object containing:  
 student\_profile: An object describing the student.  
 - id: string  
 - mastered\_topics: string[] (e.g., ["Basic Arithmetic", "Fractions"])  
 - learning\_goals: string[] (e.g., ["Introduction to Algebra", "Geometry Fundamentals"])  
 assessment\_request: An object describing the desired assessment.  
 - max\_total\_time\_minutes: integer  
 - pedagogical\_strategy: string (e.g., "REVIEW", "NEW\_TOPIC\_INTRODUCTION", "CHALLENGE")

The Orchestration Flow:  
- The /generate endpoint receives the request.  
- It invokes the Planner.  
- The Planner analyzes the student\_profile and assessment\_request to produce a detailed AssessmentPlan.  
- The endpoint then passes this AssessmentPlan to the Executor.  
- The Executor queries the problem database and assembles the GeneratedAssessment.  
- The final API response combines all of this information into a comprehensive JSON object.

## Expected API JSON Response

The API should return a structured JSON object that includes the assessment ID, planner output with reasoning log and assessment plan, and executor output with selected problems and metadata such as estimated time and constraints.

## Data

You are provided with ProblemSet.json. You should implement CRUD APIs (GET, POST, PUT, DELETE /api/problems) to manage this data store. The system should be efficient enough to handle a database with tens of thousands of problems. A MathProblem has: id, text, topic, difficulty (1-5), estimated\_time\_to\_solve\_minutes.

## Key Deliverables

- A functional Web API implementing the specified endpoints.  
- Source code for your solution.  
- A README.md file that is critically important. In it, you should explain:  
 \* Your overall architectural design and the rationale behind it.  
 \* The "contract" (AssessmentPlan structure) you designed between your Planner and Executor and why you chose that structure.  
 \* How your design allows for future extension (e.g., plugging in a real LLM, adding new problem types, supporting more complex pedagogical strategies).  
 \* Decisions you made regarding data access and potential scaling bottlenecks.