# **AI-Assisted Scoring Algorithm Refactoring Assignment**

## **Background**

You've joined Serand as a Java developer, and your first task is to refactor a critical but messy service. The SurveyResponseService currently handles too many responsibilities - it processes survey responses, calculates individual question scores, aggregates pillar scores, and computes final application scores all in one massive method.

Your task is to refactor this into a clean, modular architecture using AI tools to assist you.

## **Current State**

The provided SurveyResponseService.java has the following issues:

* The processSurveyResponse method is 400+ lines long
* Scoring logic is tightly coupled with data persistence
* No separation between different scoring strategies
* Difficult to test individual components
* Mixed responsibilities (scoring, aggregation, saving, external API calls)

## **Your Mission**

Refactor the scoring logic into two new services:

### **1. CentralScoringEngine**

Responsible for calculating individual question scores:

* Extract all question scoring logic from processSurveyResponse
* Support different question types (multiple choice, free text, coding)
* Make scoring strategies pluggable
* Return scores without side effects

### **2. ScoreCompositionService**

Responsible for aggregating scores and calculating final results:

* Handle both standard and dynamic pillar aggregation
* Calculate weighted overall scores
* Compose CV scores with survey scores
* Update application with final scores

## **Requirements**

### **Technical Requirements**

Create CentralScoringEngine.java with:  
  
 public interface CentralScoringEngine {

double calculateQuestionScore(SurveyResponseAnswer answer, Question question,

Company company, String surveyName);

Map<String, Double> calculateAllQuestionScores(SurveyResponse response,

Survey survey, Company company);

}

Create ScoreCompositionService.java with:  
  
 public interface ScoreCompositionService {

Scores aggregatePillarScores(Map<String, Map<String, SurveyResponseAnswer>> answers,

Company company, Pillars pillars);

double calculateOverallScore(Scores scores, double cvScore);

Application updateApplicationScores(Application application, Scores scores,

double cvScore);

}

1. Refactor SurveyResponseService to:  
   * Use dependency injection for the new services
   * Orchestrate the scoring flow without containing scoring logic
   * Keep the method under 100 lines
   * Maintain all existing functionality

### **AI Usage Requirements**

You MUST use AI tools (Claude, Cursor, GitHub Copilot, etc.) to:

1. Analyze the existing code structure
2. Generate the new service interfaces and implementations
3. Create comprehensive unit tests
4. Document your refactoring approach

### **Deliverables**

1. **Code Files**:  
   * CentralScoringEngine.java (interface)
   * CentralScoringEngineImpl.java (implementation)
   * ScoreCompositionService.java (interface)
   * ScoreCompositionServiceImpl.java (implementation)
   * SurveyResponseService.java (refactored version)
   * At least one unit test class
2. **AI Usage Documentation** (REQUIRED): Create a AI\_USAGE.md file documenting:  
   * Which AI tools you used
   * 3-5 example prompts that were most helpful
   * How AI accelerated your development
   * Any AI-generated code you had to modify and why
3. **Brief Design Document** (1 page max):  
   * Explain your refactoring approach
   * Describe how you separated concerns
   * List any assumptions made
   * Identify any remaining technical debt

## **Evaluation Criteria**

### **Code Quality (40%)**

* Clean separation of concerns
* Proper use of Spring patterns
* Testable, modular design
* Maintains existing functionality

### **AI Tool Proficiency (40%)**

* Quality of AI prompts used
* Effective use of AI for refactoring
* Documentation of AI workflow
* Strategic AI vs manual coding decisions

### **Problem Solving (20%)**

* Understanding of the domain
* Handling edge cases
* Design decisions and trade-offs
* Time management

## **Hints & Tips**

1. Don't try to fix everything - focus on extracting scoring logic

## 

## **Submission Instructions**

1. Create a GitHub repository with your solution
2. Include all code files and documentation
3. Share your AI conversation history (screenshots or export)
4. Submit the repository link

## **Important Notes**

* You're not expected to achieve 100% perfection in 1.5 hours
* Focus on demonstrating your refactoring skills and AI tool usage
* The code should compile and maintain core functionality
* Document any parts you would improve given more time

Good luck! Show us how AI tools can accelerate quality refactoring work.