# **Starter Project Structure**

## **Project Setup**

Create a Spring Boot project with these dependencies:

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

## **Simplified Model Classes**

Place these in src/main/java/com/serand/assessment/model/:

### **Core Models**

// Question.java

public class Question {

private String id;

private String reference;

private String question;

private String[] availableAnswers;

private String[] correctAnswers;

private boolean isGemini;

// getters/setters

}

// SurveyResponseAnswer.java

public class SurveyResponseAnswer {

private String questionId;

private String reference;

private String type; // "multipleChoice", "text", "coding"

private String[] arrayAnswer;

private String stringAnswer;

private int intAnswer;

private double questionScore;

private String scoreExplanation;

// getters/setters

}

// Scores.java

public class Scores {

private double values;

private double culture;

private double mindset;

private double growthMindset;

private WorkLife workLife;

private Map<String, Double> customPillarScores = new HashMap<>();

private Map<String, String[]> customPillarAnswers = new HashMap<>();

private Weightings weightings;

private Pillars pillars;

public double getOverallScore() {

// Calculate based on weightings or pillars

return 0.0; // Implement this

}

// getters/setters

}

// Weightings.java

public class Weightings {

private double values = 0.25;

private double culture = 0.25;

private double mindset = 0.25;

private double workLife = 0.25;

// getters/setters

}

// Pillars.java

public class Pillars {

private Map<String, Double> pillars = new HashMap<>();

// getters/setters

}

// Company.java

public class Company {

private String id;

private List<ValuesAnswer> valuesAnswers;

private String[] culture;

private WorkLifeBenefitsImpactDTO workLifeBenefitsImpact;

private Weightings weightings;

private Pillars pillars;

// getters/setters

}

## **Service Interfaces to Implement**

// GeminiService.java (Mock implementation provided)

@Service

public class GeminiService {

public String getSurveyAnswerRelevanceScore(String answer, String question, String surveyName) {

// Mock implementation - returns score and explanation

double score = Math.random(); // 0-1

return score + "-AI generated explanation for the score";

}

public void generateCandidateFeedback(Candidate candidate, Survey survey) {

// Mock implementation

}

}

// ApplicationTrackingService.java (Mock implementation provided)

@Service

public class ApplicationTrackingService {

public void pushingScoreToAts(Application application) {

// Mock implementation

System.out.println("Pushing scores to ATS for application: " + application.getId());

}

}

## **Expected Directory Structure**

src/

├── main/

│ └── java/

│ └── com/

│ └── serand/

│ └── assessment/

│ ├── model/

│ │ ├── Question.java

│ │ ├── SurveyResponseAnswer.java

│ │ ├── Scores.java

│ │ ├── Company.java

│ │ └── ... (other models)

│ ├── service/

│ │ ├── SurveyResponseService.java (provided messy version)

│ │ ├── CentralScoringEngine.java (to create)

│ │ ├── CentralScoringEngineImpl.java (to create)

│ │ ├── ScoreCompositionService.java (to create)

│ │ ├── ScoreCompositionServiceImpl.java (to create)

│ │ └── ... (mock services)

│ └── util/

│ └── ValueAnswerUtility.java

└── test/

└── java/

└── com/

└── serand/

└── assessment/

└── service/

└── CentralScoringEngineTest.java (to create)

## **Notes for Candidates**

1. Not all model fields are included - focus on what's needed for scoring
2. Mock implementations are provided for external services (Gemini, ATS)
3. You can make reasonable assumptions about missing methods
4. Focus on the refactoring task, not on making everything compile perfectly
5. Use AI to help understand the relationships between models