Compare JD and resume: API 1

1. Purpose of the Prompt

The prompt is designed to analyze a candidate's resume **against** a given **job description (JD)** and return a **strict JSON response** that evaluates the match. It ensures consistency and machine-readability for automated processing.

---

2. Structure of the Prompt

The prompt is structured into two main message roles:

1. **System Role** – Defines the rules and response format.
2. **User Role** – Provides the job description and extracted resume text for evaluation.

2.1. System Role

The **system message** acts as an instruction set for the AI. It dictates:

- **Output Format** – Must be in **strict JSON** with predefined keys.
- **Rules** – No extra text, only structured JSON.
- **Scoring System** – Numbers should stay **within 0-100**.
- **Candidate Decision** – `"yes"` or `"no"` (no other responses allowed).

2.2. User Role

The **user message** supplies:

- **Job Description (JD)**
- **Extracted Resume Text**

The AI then **compares** both and generates the evaluation.

---

3. Breakdown of the Expected JSON Response

3.1. Candidate Information (`candidate`)

This section summarizes the candidate's profile:

- `"about"` → A **brief** overview of the candidate.
- `"skills"` → A **list** of the candidate's skills.
- `"short_description"` → **Three points** describing the candidate's expertise.
- `"other_summary"` → Additional details:
  - `"experience"` → Years of experience.
  - `"preferred_location"` → Candidate's preferred job location.
  - `"current_company"` → The candidate's current employer.

3.2. Evaluation (`evaluation`)

This section scores the candidate **against the JD**:

- `"title"` → Label for evaluation.
- `"scores"` → Numerical **0-100** scores for:
  - `"Overall"` → General suitability.
  - `"Skill"` → Match of required skills.
  - `"Experience"` → Years of experience vs. JD.
  - `"Others"` → Any additional evaluation criteria.
- `"evaluation_reason"` → **Three reasons** explaining the scores.

3.3. Final Decision (`can_we_take_this_candidate`)

- Either `"yes"` or `"no"`.
- Based on skill, experience, and JD match.

---

4. Rules for Generating the Response

- **Strict JSON Format** → No extra explanations.
- **Missing Values** → Use `null` or `""`.
- **Scoring** → Always between **0-100**.
- **Final Decision** → `"yes"` or `"no"` (no other responses).

---

5. Example Responses

Example 1: Strong Match

Job Description:

Software Engineer (React, Node.js)

Resume:

- **5 years** experience
- **React, Node.js, MongoDB, AWS**

Generated JSON:


```
{
  "candidate": {
    "about": "Experienced full-stack developer with expertise in React and Node.js.",
    "skills": ["React", "Node.js", "MongoDB", "AWS"],
    "short_description": [
```

```
      "Proficient in full-stack development.",
      "Experience in building scalable applications.",
      "Strong expertise in React and Node.js."
    ],
    "other_summary": {
      "experience": "5 Years",
      "preferred_location": "",
      "current_company": ""
    },
    "evaluation": {
      "title": "Evaluation based on JD and Resume",
      "scores": {
        "Overall": 85,
        "Skill": 90,
        "Experience": 80,
        "Others": 75
      },
      "evaluation_reason": [
        "Strong match for required skills.",
        "Relevant industry experience.",
        "Good expertise in supporting technologies."
      ]
    },
    "can_we_take_this_candidate": "yes"
  }
}
```

Example 2: Weak Match

Job Description:

Data Analyst (SQL, Python, Power BI)

Resume:

- **2 years** experience
- **Excel, Python** (missing Power BI and SQL)

Generated JSON:


```
{
  "candidate": {
```

```
    "about": "Data analyst with experience in Python and Excel but lacking Power BI skills.",
    "skills": ["Excel", "Python"],
    "short_description": [
      "Strong data analysis skills.",
      "Experience in Python for data manipulation.",
      "Lacks Power BI expertise."
    ],
    "other_summary": {
      "experience": "2 Years",
      "preferred_location": "",
      "current_company": ""
    },
    "evaluation": {
      "title": "Evaluation based on JD and Resume",
      "scores": {
        "Overall": 60,
        "Skill": 50,
        "Experience": 40,
        "Others": 70
      },
      "evaluation_reason": [
        "Missing Power BI expertise.",
        "Limited experience compared to requirements.",
        "Some overlap with Python knowledge."
      ]
    },
    "can_we_take_this_candidate": "no"
  }
}
```


---

6. Conclusion

This **structured prompt** ensures that AI:

✔ **Extracts** key resume details.
✔ **Compares** them against the JD.
✔ **Scores** the candidate fairly.
✔ **Returns** a **consistent JSON output** for further processing.

Generate Preference based Questions: API 2

1. Purpose of the Prompt

The prompt is designed to **generate structured interview questions** tailored to a specific **job description (JD)** and **candidate's resume**. It ensures:

- **Relevance** → Questions align with the job role and candidate's skills.
- **Diversity** → Questions cover **technical, behavioral, and situational** aspects.
- **Structured Difficulty Levels** → Each category has **easy, medium, and hard** questions.
- **Two Sets of Questions** → Ensures variety for multiple interview rounds.

---

2. Structure of the Prompt

The prompt consists of **two main message roles**:

1. **System Role** – Defines the objective, methodology, and output format.
2. **User Role** – Supplies the job description and extracted resume text.

2.1. System Role

This section acts as an instruction set for the AI, ensuring it follows a strict format.

Key Aspects Defined in the System Role:

- **Objective** → Generate **interview questions** based on the **job description** and **resume**.

- **Methodology** → AI analyzes both sources to create **relevant and structured** questions.
- **Output Format** →
    - Returns an **array of objects** (each object represents a set of questions).
    - **Two sets of questions** (to allow variety in the interview process).
    - Each set contains **three categories**:
        - **Technical**
        - **Behavioral**
        - **Situational**
    - Each category has **three difficulty levels**:
        - **Easy**
        - **Medium**
        - **Hard**
- **Rules & Clarity** → Each question must be clear and aligned with the candidate's skills.

2.2. User Role

The **user message** provides:

- **Job Description (JD)** → Specifies required skills, experience, and role expectations.
- **Extracted Resume Text** → Candidate's actual experience, skills, and background.

The AI then **compares** both and formulates tailored interview questions.

---

3. Breakdown of the Expected JSON Response

The output is a **structured JSON array** containing **two sets** of interview questions.

3.1. Overall Structure

```
[
  {
    "set": 1,
    "categories": [
      {
        "category": "Technical",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      },
      {
        "category": "Behavioral",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      },
      {
        "category": "Situational",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      }
    ]
  },
  {
    "set": 2,
    "categories": [
      {
        "category": "Technical",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      },
```

```
      {
        "category": "Behavioral",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      },
      {
        "category": "Situational",
        "questions": {
          "easy": ["...", "...", "..."],
          "medium": ["...", "...", "..."],
          "hard": ["...", "...", "..."]
        }
      }
    ]
  }
]
```

3.2. Categories & Difficulty Levels

Each **set** contains **three categories** with **three difficulty levels**:

1️⃣ Technical Questions

Focuses on **technical knowledge, coding, and problem-solving**.
Example:

```
{
  "category": "Technical",
  "questions": {
    "easy": [
      "Can you explain the concept of a promise in JavaScript?",
      "How do you handle errors in a React application?",
      "Can you describe your experience with WebSockets?"
    ],
    "medium": [
      "Can you describe your experience with GraphQL, and how you've used it in previous projects?",
      "How do you optimize the performance of a GraphQL API?",
      "Can you explain the concept of a message queue, and how you've implemented it in a previous project?"
```

```
    ],
    "hard": [
      "Can you describe your experience with cloud computing, and how you've used it in previous projects?",
      "How do you handle security and compliance in a cloud-based architecture?",
      "Can you explain the concept of a container orchestration tool, and how you've used it in a previous project?"
    ]
  }
}
```

2️⃣ Behavioral Questions

Assesses **teamwork, communication, and adaptability**.
Example:

```
{
  "category": "Behavioral",
  "questions": {
    "easy": [
      "Can you tell me about a time when you had to work with a cross-functional team?",
      "How do you handle a situation where you're not sure about the technical requirements of a project?",
      "Can you describe your experience with agile development methodologies?"
    ],
    "medium": [
      "Can you tell me about a project you worked on that involved a significant amount of collaboration with other teams?",
      "How do you handle a situation where you're working on a team and a team member is not communicating effectively?",
      "Can you describe your experience with conflict resolution, and how you've handled conflicts in the past?"
    ],
    "hard": [
      "Can you tell me about a time when you had to make a difficult technical decision, and how you approached it?",
      "How do you handle a situation where you're working on a team and there are conflicting opinions on how to approach a technical problem?",
      "Can you describe your experience with technical leadership, and how you've mentored junior team members?"
    ]
  }
}
```

3️⃣ **Situational Questions**

Tests **decision-making, problem-solving, and adaptability**.
Example:

```
{
  "category": "Situational",
  "questions": {
    "easy": [
      "If you were given a new project with a tight deadline, how would you approach it?",
      "If you encountered a technical issue that you couldn't solve, what would you do?",
      "If you were working on a team and a team member was struggling with their tasks, what would you do to help?"
    ],
    "medium": [
      "If you were given a project with multiple stakeholders, how would you manage their expectations and priorities?",
      "If you encountered a conflict between two team members, how would you resolve it?",
      "If you were working on a project and realized that the requirements had changed, how would you adapt to the new requirements?"
    ],
    "hard": [
      "If you were given a project with a large and complex codebase, how would you approach it?",
      "If you encountered a technical issue that required a significant amount of time and resources to solve, how would you prioritize it?",
      "If you were working on a team and the project was at risk of failing, what would you do to turn it around?"
    ]
  }
}
```

---

4. Key Features & Rules

✅ **Strict JSON Format** → No extra text, only structured JSON.
✅ **Two Sets of Questions** → Ensures variety for different interview rounds.
✅ **Three Categories** → Covers Technical, Behavioral, and Situational aspects.
✅ **Three Difficulty Levels** → Ensures progressive challenge.
✅ **Custom Questions** → AI tailors questions to **job description** & **resume**.

---

5. Conclusion

This **structured prompt** ensures that AI:
✔ **Extracts** job-specific information.
✔ **Generates relevant interview questions** tailored to the candidate.
✔ **Formats the output correctly** for easy processing.

Personalised Questions Chatbot: API 3

1️⃣ Purpose of the Prompt

This AI-driven interviewer is designed to:

- **Generate structured, technical interview questions** for a given job role.
- **Ensure diversity in topics** by avoiding repetitive questions.
- **Focus on relevant skill sets** mentioned in the **candidate's resume** and **job description**.
- **Prevent redundancy** by filtering out previously asked questions.
- **Maintain a structured format** for easy processing.

2️⃣ Structure of the Prompt

2.1. System Message

The system message defines the AI's role:

{ "role": "system", "content": "You are a structured AI interviewer." }

✅ Ensures that AI behaves as an **interviewer**, focusing only on generating **structured, job-relevant questions**.

2.2. User Message (Dynamic Prompt)

The user message contains the main instructions.

const prompt = `
// You are an AI interviewer generating **${difficulty} ${category} interview questions** based on the job description and candidate's resume. Ask more questions related to the skillset mentioned in the Resume/user CV, ensuring that each question covers a distinct concept.

- `${difficulty}` → Specifies the **difficulty level** (easy, medium, or hard).
- `${category}` → Defines the **question category** (e.g., Technical, Behavioral).
- **Ensures topic diversity** → Each question should introduce **a new concept** rather than repeating previous ones.

2.3. Rules for Generating Questions

The AI follows these rules while generating questions:

1️⃣ **Avoid Repetition:**

// DO NOT repeat past questions and answers.

✅ Ensures **uniqueness** by avoiding questions that have already been asked.

2️⃣ **Ensure Diversity in Concepts:**

// For example, if a question is asked about jet engine fuel injection, the next question should focus on a different aspect, such as how an engine carburetor is made.

✅ If a **previous question** focused on **React's useState**, the next one should explore **Redux or Context API** instead.

3️⃣ **Keep Questions Technical & Core Concept-Based:**

// Ensure that questions are technical and core concept-based rather than generic.

✅ **Avoids generic or surface-level questions**, ensuring depth.

4️⃣ **Focus on Job Relevance:**

// If the candidate is from a software background, ask programming-related questions on topics such as state management, differences between useRef() and useMemo(), etc.

✅ **Tailors** questions based on the **candidate's industry & expertise**.

3️⃣ Context Variables Used in the Prompt

The following **three dynamic variables** provide context for generating questions:

- **Job Description:**
- **Job Description:** "${jobDescription}"

✅ **Ensures** questions align with the **requirements of the job role**.

- **Candidate's Resume:**
- **Candidate's Resume:** "${extractedText}"

✅ **Ensures** questions are relevant to the **candidate's skill set & experience**.

- **Past Questions & Answers:**
- **Past Questions & Answers:** ${formattedHistory}

✅ **Prevents the AI from asking duplicate questions**.

4️⃣ Expected Output Format

The AI must return the question in **strict JSON format**, following this structure:

```
{
  "Question": "Can you explain the difference between using 'useState' and Redux for state management in React and provide scenarios where each is most appropriate?",
  "EstimatedTime": "3 minutes"
}
```

✅ **Strict structure** makes it easy to use the output in an application.

Example Output

```
{
  "Question": "How does garbage collection work in JavaScript, and what are the different types of garbage collection techniques?",
  "EstimatedTime": "2 minutes"
}
```

5️⃣ Key Constraints & Rules

🔴 **Rules to Follow:**
1️⃣ **DO NOT repeat past questions.**
2️⃣ **Focus on job relevance.**
3️⃣ **Ask only one question at a time.**
4️⃣ **Ensure each question covers a new concept.**
5️⃣ **Ignore irrelevant user responses.**
6️⃣ **Return only the question in the specified format.**

6️⃣ OpenAI API Call Execution

Finally, the prompt is sent to OpenAI's API using **GPT-4o-mini**:

```
const stream = await openai.chat.completions.create({
  model: "gpt-4o-mini",
  messages: [
    { role: "system", content: "You are a structured AI interviewer." },
    { role: "user", content: prompt }
  ],
  stream: true,
});
```

Explanation of API Call:

1️⃣ **Model Used:** `"gpt-4o-mini"` → Ensures fast and cost-efficient responses.
2️⃣ **Streaming Enabled:** `"stream: true"` → Provides real-time question generation.
3️⃣ **Two Message Roles:**

- `{ role: "system", content: "You are a structured AI interviewer." }`
- `{ role: "user", content: prompt }`

✅ The **user's request** (including the job description, resume, and past questions) is passed to GPT, which then **generates a unique, structured interview question**.

7️⃣ Summary

🔹 **What this prompt does:**
✔ Generates **structured technical interview questions**.
✔ Ensures **each question is unique** and **covers a new concept**.
✔ Filters **past questions** to prevent repetition.
✔ Adapts to the **candidate's resume & job description**.
✔ Returns output in **strict JSON format**.

🔹 **Why this is useful:**
✔ Helps automate **customized, AI-driven interviews**.
✔ Ensures **high-quality, relevant** questions.
✔ Can be integrated into **chatbots, interview tools, or HR systems**.

Predict the score: API 4

Detailed Explanation of `getOpenAIChatCompletion(userResponses)`

This function **evaluates a candidate's responses** based on predefined scoring criteria and returns a **structured JSON object** containing scores, feedback, and suggestions for improvement.

1️⃣ Function Overview

async function getOpenAIChatCompletion(userResponses) { ... }

- **Purpose**: This function sends **candidate responses** to OpenAI's GPT-4o-mini and retrieves an **evaluation** based on multiple criteria.
- **Returns**: A structured **JSON object** containing **scores, feedback, and suggested improvements**.
- **Key Features**:
  ✔️ Uses OpenAI to analyze **technical, communication, problem-solving, and soft skills**.
  ✔️ Ensures **structured, consistent output** in JSON format.
  ✔️ Parses and validates AI-generated output to avoid errors.

2️⃣ Prompt Breakdown

🔹 Constructing the Evaluation Prompt

const prompt = `You are an AI evaluator analyzing candidate responses based on various scoring criteria.

- This defines the **AI's role** as an evaluator that scores a candidate's responses based on predefined **scoring criteria**.

🔹 Including Candidate Responses

🔹 **Candidate Responses:**
${JSON.stringify(userResponses, null, 2)}

- The AI receives the **candidate's responses** as JSON, allowing it to analyze the exact answers given during an interview.

🔹 Defining the Scoring Criteria

Scoring Criteria:
- **Technical Acumen**: Evaluate the candidate's technical knowledge demonstrated in their responses.
- **Communication Skills**: Assess clarity, coherence, and effectiveness in conveying ideas.
- **Responsiveness & Agility**: Determine how promptly and thoughtfully the candidate responds. Use the difference between answered time and question asked time for delay of user.
- **Problem-Solving & Adaptability**: Analyze ability to handle follow-up questions and clarifications.
- **Cultural Fit & Soft Skills**: Evaluate interpersonal communication and potential fit.

The AI is instructed to evaluate responses using **five main criteria**:

1️⃣ **Technical Acumen** → How well the candidate understands the technical concepts.
2️⃣ **Communication Skills** → Clarity, coherence, and articulation of ideas.
3️⃣ **Responsiveness & Agility** → Measures promptness in answering and thoughtfulness.
4️⃣ **Problem-Solving & Adaptability** → Ability to handle complex problems and follow-ups.
5️⃣ **Cultural Fit & Soft Skills** → Measures interpersonal communication and team fit.

✅ **Ensures structured evaluation across multiple dimensions.**

🔹 Overall Score Calculation

- **The overall should be the sum of all other**:
  - "Technical"

- "Communication"
- "Responsiveness"
- "ProblemSolving"
- "SoftSkills"
- "Responded"

- The **Overall** score is derived by summing the scores of all individual criteria.
- **Ensures consistency** in scoring methodology.

🔹 Critical Note for JSON Output Format

CRITICAL NOTE : THE RESPONSE SHOULD BE IN THE FOLLOWING FORMAT ONLY.

- This ensures the AI **strictly adheres to a structured JSON response format**, preventing unstructured or incomplete outputs.

🔹 Expected JSON Output Format

```
{
  "OverAll": number,
  "Technical": number,
  "Communication": number,
  "Responsiveness": number,
  "ProblemSolving": number,
  "SoftSkills": number,
  "Responded": number,
  "feedback": [
    { "category": "Technical Acumen", "comment": "string" },
    { "category": "Communication Skills", "comment": "string" },
    { "category": "Responsiveness & Agility", "comment": "string" },
    { "category": "Problem-Solving & Adaptability", "comment": "string" },
    { "category": "Cultural Fit & Soft Skills", "comment": "string" }
  ],
  "suggestedImprovements": ["string"]
}
```

This **ensures a structured response** with:1️⃣ **Numerical scores** for different criteria.
2️⃣ **Detailed feedback** on each category.
3️⃣ **List of improvement suggestions**.

✅ **Consistent formatting allows easy integration into dashboards or reporting tools.**

3️⃣ OpenAI API Call

The function sends the prompt to **GPT-4o-mini** using OpenAI's API:

```
const response = await openai.chat.completions.create({
  model: "gpt-4o-mini",
  messages: [
    { role: "system", content: "You are an AI interviewer and evaluator." },
    { role: "user", content: prompt },
  ],
  temperature: 0.7,
});
```

📌 Key Components of API Call

1️⃣ `model: "gpt-4o-mini"` → Uses GPT-4o-mini for cost-effective and fast response generation.
2️⃣ `messages` →

- `{ role: "system", content: "You are an AI interviewer and evaluator." }`
- `{ role: "user", content: prompt }`
  ✅ Provides **clear role instructions** to the AI.
  3️⃣ `temperature: 0.7` → Adds some **creativity** while ensuring structured and reliable responses.

4️⃣ Parsing and Error Handling

The API response is parsed into **JSON format**, with error handling to prevent failures.

```
try {
  return JSON.parse(response.choices[0]?.message?.content || "{}") || {};
} catch (error) {
  console.error("Failed to parse OpenAI response:", error);
  return {};
}
```

📌 What This Does

1️⃣ **Attempts to parse AI response as JSON** using `JSON.parse()`.
2️⃣ **Handles parsing errors** → If parsing fails, an empty object `{}` is returned.
3️⃣ **Prevents application crashes** if OpenAI returns invalid data.

✅ **Ensures robustness and reliability** in handling AI-generated responses.

5️⃣ Example Input & Output

📌 Example Input (Candidate Responses)

```
{
  "response1": "I have used Redux for state management in large applications.",
  "response2": "I believe communication is key when working in a team."
}
```

📌 Example Output (AI-Generated Evaluation)

```
{
  "OverAll": 85,
  "Technical": 90,
  "Communication": 80,
  "Responsiveness": 75,
  "ProblemSolving": 85,
  "SoftSkills": 80,
  "Responded": 70,
  "feedback": [
    { "category": "Technical Acumen", "comment": "Strong understanding of Redux for state
management." },
```

{ "category": "Communication Skills", "comment": "Clear explanation but could provide more structured examples." },
{ "category": "Responsiveness & Agility", "comment": "Responses are thoughtful but slightly delayed." },
{ "category": "Problem-Solving & Adaptability", "comment": "Good ability to explain but lacks deeper problem-solving scenarios." },
{ "category": "Cultural Fit & Soft Skills", "comment": "Shows an understanding of teamwork and communication." }
],
"suggestedImprovements": [
"Provide more in-depth technical examples.",
"Try answering follow-up questions more quickly."
]
}

📌 Breakdown of Output

✅ **Numeric Scores** (Evaluates performance in each area).
✅ **Feedback Comments** (Explains the reason behind each score).
✅ **Suggested Improvements** (Helps candidates improve).

6️⃣ Summary

✅ What This Function Does

✔ **Evaluates** candidate responses based on **technical, communication, and problem-solving skills**.
✔ **Prevents redundancy** with structured **JSON formatting**.
✔ **Ensures robustness** with error handling and validation.
✔ **Uses AI to provide human-like evaluation** and **actionable feedback**.

🚀 Why This is Useful

- 🔹 **Automates candidate evaluations** for HR teams.
- 🔹 **Provides objective feedback** to help candidates improve.
- 🔹 **Can be integrated** into interview platforms, hiring tools, or learning dashboards.