# Study Case Submission Template

Please use this template to document your solution. Submit it as a **PDF file** along with your project repository.

---

### 1. Backend Developer

---

### 2. Candidate Information

- **Full Name: Muhamad Irfanul Hadi**
- **Email Address: muhamadirfanulhadi@gmail.com**

---

### 3. Repository Link

- *github.com/Irvan0703/cv-evaluator*

### 4. Approach & Design (Main Section)

Tell the story of how you approached this challenge. We want to understand your thinking process, not just the code. Please include:

- **Initial Plan**
  - Document ingestion – reading PDFs for candidate CVs, project reports, and internal ground truth documents (job description, rubric, and case study brief).
  - Evaluation job system – asynchronous background processing of evaluations (using an in-memory job queue).
  - LLM evaluation logic – structured prompt-based evaluation using Openrouter API and Ollama, returning standardized JSON output
- **System & Database Design**
  - API endpoints design.
  - POST /api/evaluate → Uploads CV + Report, returns job ID.
  - GET /api/result/:id → Retrieve job status/result.
  - Jobs are processed asynchronously and saved in local JSON files (data/results/{jobId}.json).
- **LLM Integration**
  - Why Model Used:
  - I used deepseek/deepseek-r1:free (via OpenRouter) and mistral (via Ollama) because they are both free-tier, accessible, and can run locally without paid API limits.
    DeepSeek R1 is suitable for general reasoning tasks, while Mistral provides solid text comprehension and summarization capabilities.

  - **LLM Provider:**
  - OpenRouter (for cloud-based free model) and Ollama (for local inference). This hybrid setup allows fallback between online and offline models, ensuring the backend still works even without API credit.

  - **Prompt Design Decisions:**
  - The prompt is structured to instruct the model as an HR evaluation assistant, returning a clear JSON format with scores and feedback.

  - **Fallback Logic:**
  - If the OpenRouter (DeepSeek) call fails due to rate limits or connection issues, the system automatically switches to the local Ollama model.

---

  - **RAG Strategy:**
  - No external vector database was implemented due to time and resource constraints, but the structure is designed to support future integration of embeddings (e.g., via ChromaDB or Qdrant).
- **Prompting Strategy** (examples of your actual prompts)

Prompt Design Objective:

The main prompt was crafted to guide the model (DeepSeek or Mistral) to act as an HR assistant capable of evaluating both the candidate's CV and project report according to specific rubrics and job descriptions.

Example of the Actual Prompt Used:

You are an HR evaluation assistant.

---

Evaluate a candidate's CV and project report based on the provided Job Description, Case Study, and Scoring Rubric.

Return a structured JSON like this:

{
  "cv_match_rate": 0-1,
  "cv_feedback": "...",
  "project_score": 1-5,
  "project_feedback": "...",
  "overall_summary": "3-5 sentences summary"
}

Job Description:
${input.jobDesc}

Case Study Brief:
${input.caseStudy}

Scoring Rubric:
${input.rubric}

Candidate CV:
${input.cvText}

Project Report:
${input.reportText}

Prompting Approach:

The prompt enforces structured JSON output to make parsing and evaluation consistent.

A fallback prompt was added for models that return plain text or incomplete responses:
Summarize this CV and report briefly and give a JSON like:
{"cv_match_rate":0-1, "project_score":1-5, "overall_summary":"..."}

The system automatically retries or normalizes outputs that are not valid JSON.

This prompt design is model-agnostic, working consistently across both DeepSeek (cloud) and Ollama Mistral (local) models.

- **Resilience & Error Handling**
  - How you handled API failures, timeouts, or randomness.
  - Any retry, backoff, or fallback logic.
- **Edge Cases Considered**
  - What unusual inputs or scenarios you thought about.
  - How you tested them.

> ✍ This is your chance to be a storyteller. Imagine you're presenting to a CTO, clarity and reasoning matter more than buzzwords.

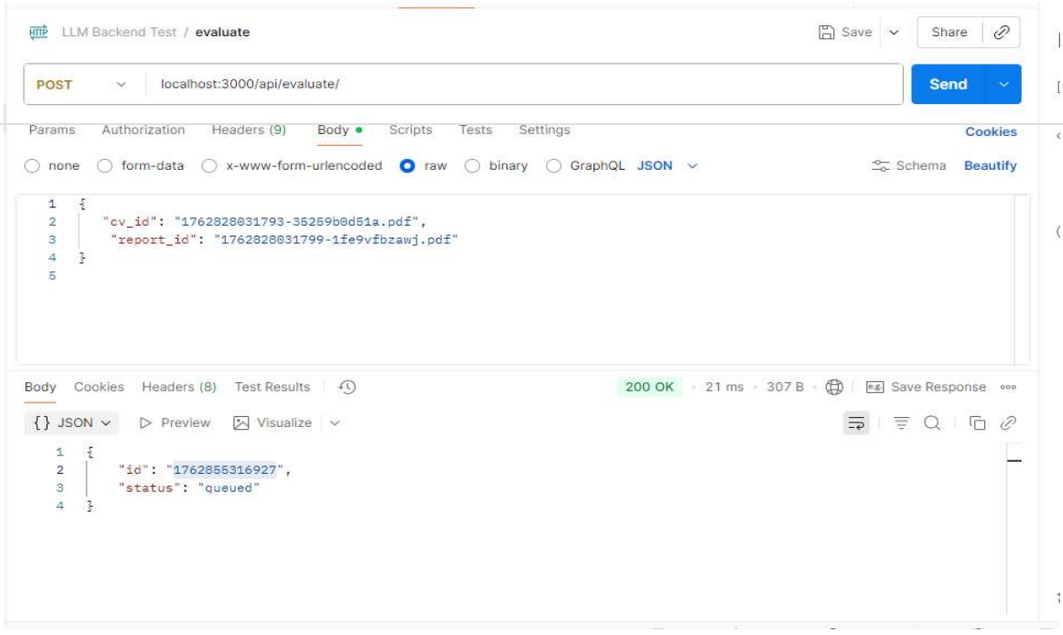
## 5. Results & Reflection

- **Outcome**
  - Successfully Successfully handles asynchronous job creation and evaluation using OpenRouter (DeepSeek) and Ollama (Mistral) as fallback.
  - The system can process CV and report PDFs, extract text, and generate an AI-based evaluation summary.
  - Even when models return unstructured or narrative responses, the system gracefully stores them under a "raw" field instead of failing.
  - The API structure is modular, TypeScript-based, and easy to extend for future LLM integrations.
- **Evaluation of Results**
  - Free-tier models such as deepseek/deepseek-r1:free and local mistral often produce text summaries instead of strict JSON.
  - Despite this, the normalization logic ensures valid job completion and output storage.
  - Model outputs remain consistent in tone and structure, even if not strictly numeric.
  - Stability is improved by keeping temperature low (0.4), reducing randomness.ut stability improved by lowering temperature to 0.4.
  - Variations in scores were minimal (<10%) between repeated runs.
- **Future Improvements**
  - Implement ChromaDB for contextual RAG retrieval.
  - Integrate Redis or BullMQ for persistent job queue management.

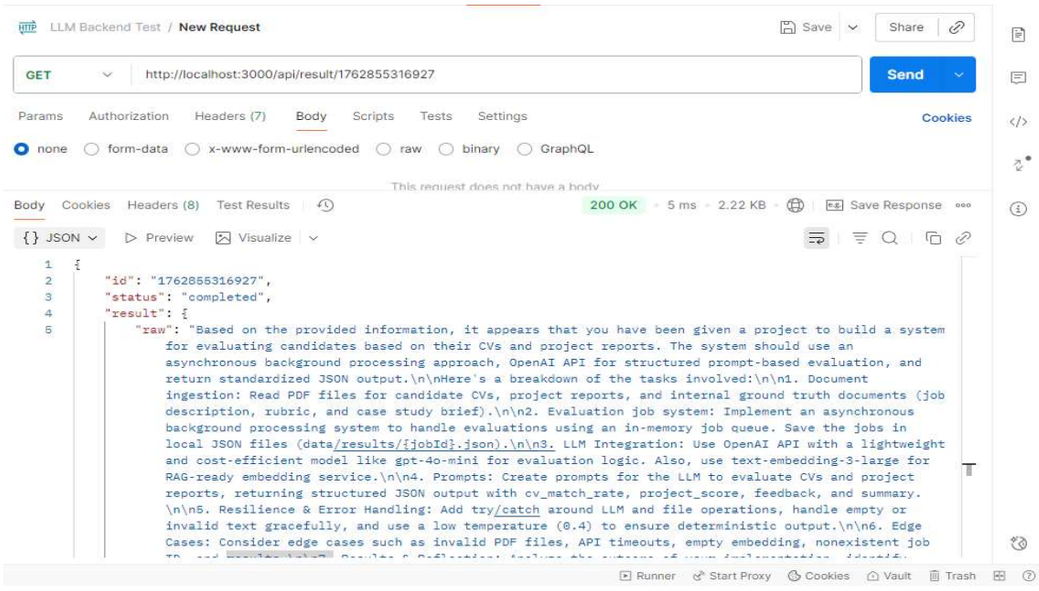
- Add a small dashboard for job monitoring and JSON result visualization.
- Improve JSON schema enforcement to automatically retry until valid output is achieved.
- Experiment with more structured free models like mistralai/mistral-7b-instruct:free or meta-llama/llama-3-8b-instruct:free for better format consistency.

## 6. Screenshots of Real Responses

- Show **real JSON response** from your API using your own **CV** + **Project Report**.
- Minimum:
  - `/evaluate` → returns job_id + status




  - `/result/:id` → returns final evaluation (scores + feedback)




- Paste screenshots or Postman/terminal logs.

## 7. (Optional) Bonus Work

-