Case Study Brief

Hello! Thank you for applying with us as a backend developer. This mini project **should be completed within 5 days after you have received this document**. Please spare your time to complete this project with the best results. We are really pleased to answer your questions if there are unclear things.

Objective

Your mission is to build a backend service that automates the initial screening of a job application. The service will receive a candidate's CV and a project report, evaluate them against a specific job description and a case study brief, and produce a structured, AI-generated evaluation report.

Core Logic & Data Flow

The system operates with a clear separation of inputs and reference documents:

Candidate-Provided Inputs (The Data to be Evaluated):

1. **Candidate CV:** The candidate's resume (PDF).
2. **Project Report:** The candidate's project report to our take-home case study (PDF)

System-Internal Documents (The "Ground Truth" for Comparison):

1. **Job Description:** A document detailing the requirements and responsibilities for the role — You can use the job description you're currently applying. This document will be used as ground truth for **Candidate CV**.
    - To make sure the vector retrieval is accurate enough, you might need to ingest a few job description documents as well.
2. **Case Study Brief:** This document. Used as ground truth for **Project Report**. (PDF)
3. **Scoring Rubric:** A predefined set of parameters for evaluating CV and Report, each has it's own documents. (PDF)

We want to see your ability to combine **backend engineering** with **AI workflows** (prompt design, LLM chaining, retrieval, resilience).

Deliverables

1. Backend Service (API endpoints)

Implement a backend service with at least the following RESTful API endpoints:

- `POST /upload`
    - Accepts multipart/form-data containing the Candidate CV and Project Report (PDF).
    - Stores these files, return each with it's own ID for later processing.
- `POST /evaluate`
    - Triggers the asynchronous AI evaluation pipeline. Receives input job title (string), and **both** document ID.
    - Immediately returns a job ID to track the evaluation process.

```
{
  "id": "456",
  "status": "queued"
}
```

- `GET /result/{id}`
    - Retrieves the status and result of an evaluation job. This endpoint should reflect the asynchronous, **multi-stage** nature of the process.
    - Possible responses:
        - While queued or processing

```
{
  "id": "456",
  "status": "queued" | "processing"
}
```

        - Once completed

```
{
  "id": "456",
  "status": "completed",
  "result": {
    "cv_match_rate": 0.82,
    "cv_feedback": "Strong in backend and cloud, limited AI integration experience...",
    "project_score": 4.5,
    "project_feedback": "Meets prompt chaining requirements, lacks error handling robustness...",
    "overall_summary": "Good candidate fit, would benefit from deeper RAG knowledge..."
  }
}
```

2. Evaluation Pipeline

Design and implement an AI-driven pipeline which will be triggered by **[POST]** `/evaluate` endpoint. Should consist these key Components:

- **RAG (Context Retrieval)**
  - Ingest all **System-Internal Documents** (Job Description, Case Study Brief, Both Scoring Rubrics) into a vector database.
  - Retrieve relevant sections and inject into prompts (e.g., "for CV scoring" vs "for project scoring").
- **Prompt Design & LLM Chaining**

  The pipeline should consists of
  - CV Evaluation
    - Parse the candidate's CV into structured data.
    - Retrieve relevant information from both Job Description and CV Scoring Rubrics.
    - Use an LLM to get these result: `cv_match_rate` & `cv_feedback`
  - Project Report Evaluation
    - Parse the candidate's Project Report into structured data.
    - Retrieve relevant information from both Case Study Brief and CV Scoring Rubrics.
    - Use an LLM to get these result: `project_score` & `project_feedback`
  - Final Analysis
    - Use a final LLM call to synthesize the outputs from previous steps into a concise `overall_summary`.
- **Long-Running Process Handling**
  - `POST /evaluate` should **not block** until LLM Chaining finishes.
  - Store task, return job ID, allow `GET /result/{id}` to check later periodically.
- **Error Handling & Randomness Control**
  - Simulate any edge cases you can think of and how well your service can handle them.
  - Simulate failures from LLM API (timeouts, rate limit).
  - Implement retries/back-off.
  - Control LLM temperature or add validation layer to keep responses stable.

---

3. Standardized Evaluation Parameters

Define at least these scoring parameters:

CV Evaluation (Match Rate)

- **Technical Skills Match** (backend, databases, APIs, cloud, AI/LLM exposure).
- **Experience Level** (years, project complexity).
- **Relevant Achievements** (impact, scale).
- **Cultural Fit** (communication, learning attitude).

Project Deliverable Evaluation

- **Correctness** (meets requirements: prompt design, chaining, RAG, handling errors).
- **Code Quality** (clean, modular, testable).
- **Resilience** (handles failures, retries).

- **Documentation** (clear README, explanation of trade-offs).
- **Creativity / Bonus** (optional improvements like authentication, deployment, dashboards).

Each parameter can be scored **1–5**, then aggregated to final score.

Requirements

- Use any backend framework (Rails, Django, Node.js, etc.).
- Use a proper LLM service (e.g., OpenAI, Gemini, or OpenRouter). There are several free LLM API providers available.
- Use a simple **vector DB** (e.g. ChromaDB, Qdrant, etc) or **RAG-as-a-service** (e.g. Ragie, S3 Vector, etc), any of your own choice.
- Provide README with run instructions + explanation of design choices.
- Provide the documents together with their ingestion scripts in the repository for reproducibility purposes.

Scoring Rubric for Case Study Evaluation

CV Match Evaluation (1–5 scale per parameter)

Parameter	Description	Scoring Guide
Technical Skills Match (Weight: 40%)	Alignment with job requirements (backend, databases, APIs, cloud, AI/LLM).	1 = Irrelevant skills, 2 = Few overlaps, 3 = Partial match, 4 = Strong match, 5 = Excellent match + AI/LLM exposure
Experience Level (Weight: 25%)	Years of experience and project complexity.	1 = <1 yr / trivial projects, 2 = 1–2 yrs, 3 = 2–3 yrs with mid-scale projects, 4 = 3–4 yrs solid track record, 5 = 5+ yrs / high-impact projects
Relevant Achievements (Weight: 20%)	Impact of past work (scaling, performance, adoption).	1 = No clear achievements, 2 = Minimal improvements, 3 = Some measurable outcomes, 4 = Significant contributions, 5 = Major measurable impact
Cultural / Collaboration Fit (Weight: 15%)	Communication, learning mindset, teamwork/leadership.	1 = Not demonstrated, 2 = Minimal, 3 = Average, 4 = Good, 5 = Excellent and well-demonstrated

Project Deliverable Evaluation (1–5 scale per parameter)

Parameter	Description	Scoring Guide
Correctness (Prompt & Chaining) (Weight: 30%)	Implements prompt design, LLM chaining, RAG context injection.	1 = Not implemented, 2 = Minimal attempt, 3 = Works partially, 4 = Works correctly, 5 = Fully correct + thoughtful
Code Quality & Structure (Weight: 25%)	Clean, modular, reusable, tested.	1 = Poor, 2 = Some structure, 3 = Decent modularity, 4 = Good structure + some tests, 5 = Excellent quality + strong tests
Resilience & Error Handling (Weight: 20%)	Handles long jobs, retries, randomness, API failures.	1 = Missing, 2 = Minimal, 3 = Partial handling, 4 = Solid handling, 5 = Robust, production-ready
Documentation & Explanation (Weight: 15%)	README clarity, setup instructions, trade-off explanations.	1 = Missing, 2 = Minimal, 3 = Adequate, 4 = Clear, 5 = Excellent + insightful
Creativity / Bonus (Weight: 10%)	Extra features beyond requirements.	1 = None, 2 = Very basic, 3 = Useful extras, 4 = Strong enhancements, 5 = Outstanding creativity

3. Overall Candidate Evaluation

- **CV Match Rate:** Weighted Average (1–5) → Convert to 0–1 decimal (×0.2).
- **Project Score:** Weighted Average (1–5)
- **Overall Summary:** Service should return 3–5 sentences (strengths, gaps, recommendations).

Study Case Submission Template

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

---

1. Title

---

2. Candidate Information

- **Full Name:**
- **Email Address:**

---

3. Repository Link

- Provide a link to your GitHub repository.
- ⚠ **Important:** Do **not** use the word *Rakamin* anywhere in your repository name, commits, or documentation. This is to reduce plagiarism risk.
- Example: *github.com/username/ai-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**
  - How you broke down the requirements.
  - Key assumptions or scope boundaries.
- **System & Database Design**
  - API endpoints design.
  - Database schema (diagram or explanation).
  - Job queue / long-running task handling.
- **LLM Integration**
  - Why you chose a specific LLM or provider.
  - Prompt design decisions.
  - Chaining logic (if any).
  - RAG (retrieval, embeddings, vector DB) strategy.
- **Prompting Strategy** (examples of your actual prompts)
- **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**
  - What worked well in your implementation?
  - What didn't work as expected?
- **Evaluation of Results**
  - If the evaluation scores/outputs were bad or inconsistent, explain why.
  - If they were good, explain what made them stable.
- **Future Improvements**
  - What would you do differently with more time?
  - What constraints (time, tools, API limits) affected your solution?

---

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

If you added extra features, describe them here.