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: LLM Integration For CV and Project Report Analyzer With RAG Implementation

---

2. Candidate Information

- **Full Name: M. Fadil Martias**
- **Email Address: fadilmartias26@gmail.com**

---

3. Repository Link

- https://github.com/fadilmartias/cv-analyzer

---

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**
  - After reviewing the requirements, I started by thinking about the tech stack that would let me move fast while keeping things robust. I decided to go with **Go** and the **Fiber framework**. Fiber is built on **fasthttp**, which is super fast, and its goroutines are perfect for handling queues and long-running tasks without blocking. Plus, being a compiled language, Go helps catch bugs early during development, so I can minimize headaches in production.

    For the database, I went with **PostgreSQL**, mainly because I'm familiar with it and it also has the **pgvector extension**, which lets me store embeddings directly in the database.

    For PDF reading, I initially tried **UniPDF**, but the free version couldn't extract text from scanned PDFs. After hitting some roadblocks, I switched to **Tesseract OCR** on my Windows machine, which worked much better for scanned documents.

    For AI/LLM, I started with **OpenRouter** because it's free and has lots of model options. But midway through, I realized OpenRouter doesn't provide embedding endpoints, so I pivoted to **Gemini by Google**, which supports embeddings and fits perfectly for RAG (Retrieval-Augmented Generation) workflows.

    The goal of this mini-project was to allow a user to **upload a CV and project report**, have an AI evaluate them asynchronously, and return detailed scores and feedback, while also leveraging vector databases for intelligent retrieval.
- **System & Database Design**

  I kept the system simple but scalable:
  - **Endpoints**:

    POST /evaluate → for uploading CV and project report

    GET /result/{id} → to fetch AI evaluation results
  - **Database**:

    evaluation_tasks: stores tasks with fields like cv, report, status, detailed scoring (cv_match_rate, cv_feedback, project_score, project_feedback, overall_summary, breakdown).

    jobs: stores job descriptions along with embeddings (title, content, embedding vector) for RAG retrieval.
  - **Long-running tasks**:

    I leveraged Go's **goroutines**. Every call to the LLM runs asynchronously, so /evaluate doesn't block.
  - **Project structure**:

    I followed **clean architecture**:

    usecase → business logic

    repository → database interaction

service → external services (like Gemini)
handler → HTTP endpoints
config → environment variables using godotenv
dto → consistent API responses
utils → helper functions like PDF extraction and format json response for all API Response

- **LLM Integration**
  - I used Gemini because it's free and supports embeddings.
  - The AI acts as a technical recruiter, evaluating both the CV and project report against job requirements.
  - I implemented RAG to fetch relevant job context for scoring, improving the relevance of AI feedback.
  - I created a small set of dummy jobs to populate embeddings and test the pipeline.
- **Prompting Strategy** (examples of your actual prompts)

```
You are an experienced technical recruiter. Analyze the following CV and Project Report against these job requirements:
(here is the context from RAG)
Return your answer STRICTLY in JSON format with this schema:
{
  "cv_match_rate": <float with 2 decimal places, range 0-1 based on cv breakdown score>,
  "cv_feedback": "<feedback about CV>",
  "project_score": <float with 2 decimal places, range 0-10 based on project breakdown score>,
  "project_feedback": "<feedback about Project Report>",
  "overall_summary": "<summary of overall impression, strengths, and areas to improve>",
 "breakdown": {
  "cv": {
  "technical_skills_match": <number 1-5, criteria: backend, databases, APIs, cloud, and AI/LLM exposure>,
  "experience_level": <number 1-5, criteria: years, project complexity>,
  "relevant_achievements": <number 1-5, criteria: impact, scale>,
  "cultural_fit": <number 1-5, criteria: communication, learning attitude>,
  },
  "project_report": {
   "correctness": <number 1-5, criteria: prompt design, chaining, RAG, handling errors>,
   "code_quality": <number 1-5, criteria: clean, modular, testable>,
   "resilience": <number 1-5, criteria: handles failures, retries>,
   "documentation": <number 1-5, criteria: clear README, explanation of trade-offs>,
   "creativity_or_bonus": <number 1-5, criteria: optional improvements like authentication, deployment, dashboards, etc.>
  }
 },

}

CV:
(extracted cv)

Report:
(extracted project_report)
```

- **Resilience & Error Handling**
  I wanted to make sure the system could handle all the usual hiccups:
  - Retry logic with **exponential backoff** for Gemini API calls
  - Timeout configurations for LLM requests

- Circuit breaker to prevent cascading failures
- Temperature set to **0.1** for consistent, low-hallucination results
- Strict JSON validation before storing results

- **Edge Cases Considered**
    - Large, encrypted, or password-protected PDFs
    - Empty or minimal content → validation on CV length
    - Memory leaks in long-running goroutines → monitor goroutine counts
    - Invalid file types → add validation for check file types
    - Rate limiting: /evaluate calls limited to 1 per 4 seconds to stay under Gemini free-tier limits
    - Integration and load testing to ensure stability

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

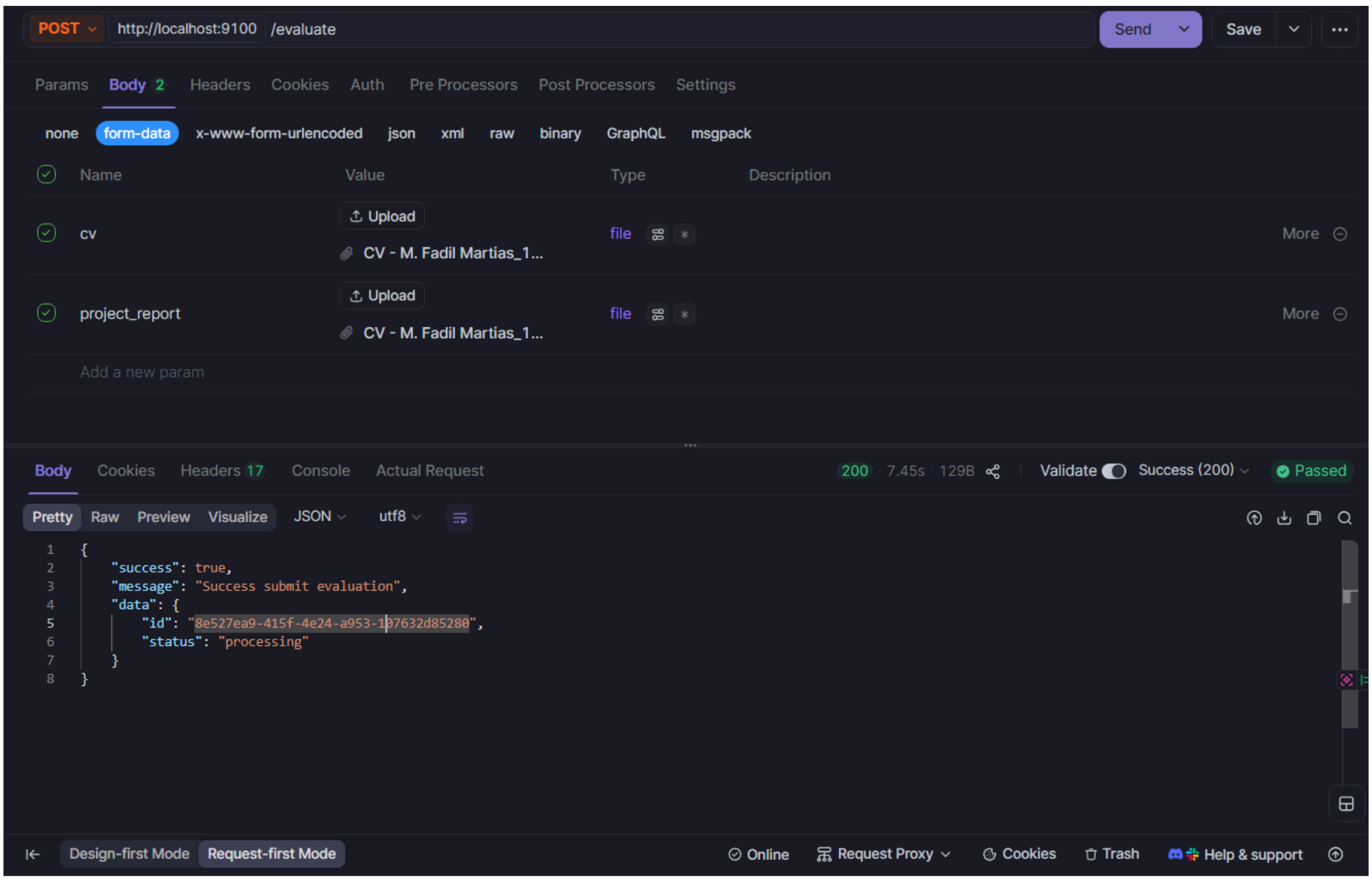
5. Results & Reflection

- **Outcome**
  - Everything worked as expected. Users can upload CVs and project reports, get async AI evaluations, and store results in Postgres with embeddings for retrieval.
- **Evaluation of Results**
  - AI outputs are consistent thanks to low temperature settings. JSON schema validation ensures structure is reliable.
- **Future Improvements**
  - Handle more error cases for maximum robustness
  - Possibly move to paid-tier LLMs or local embeddings for higher throughput
  - Improve PDF OCR accuracy for tricky scans

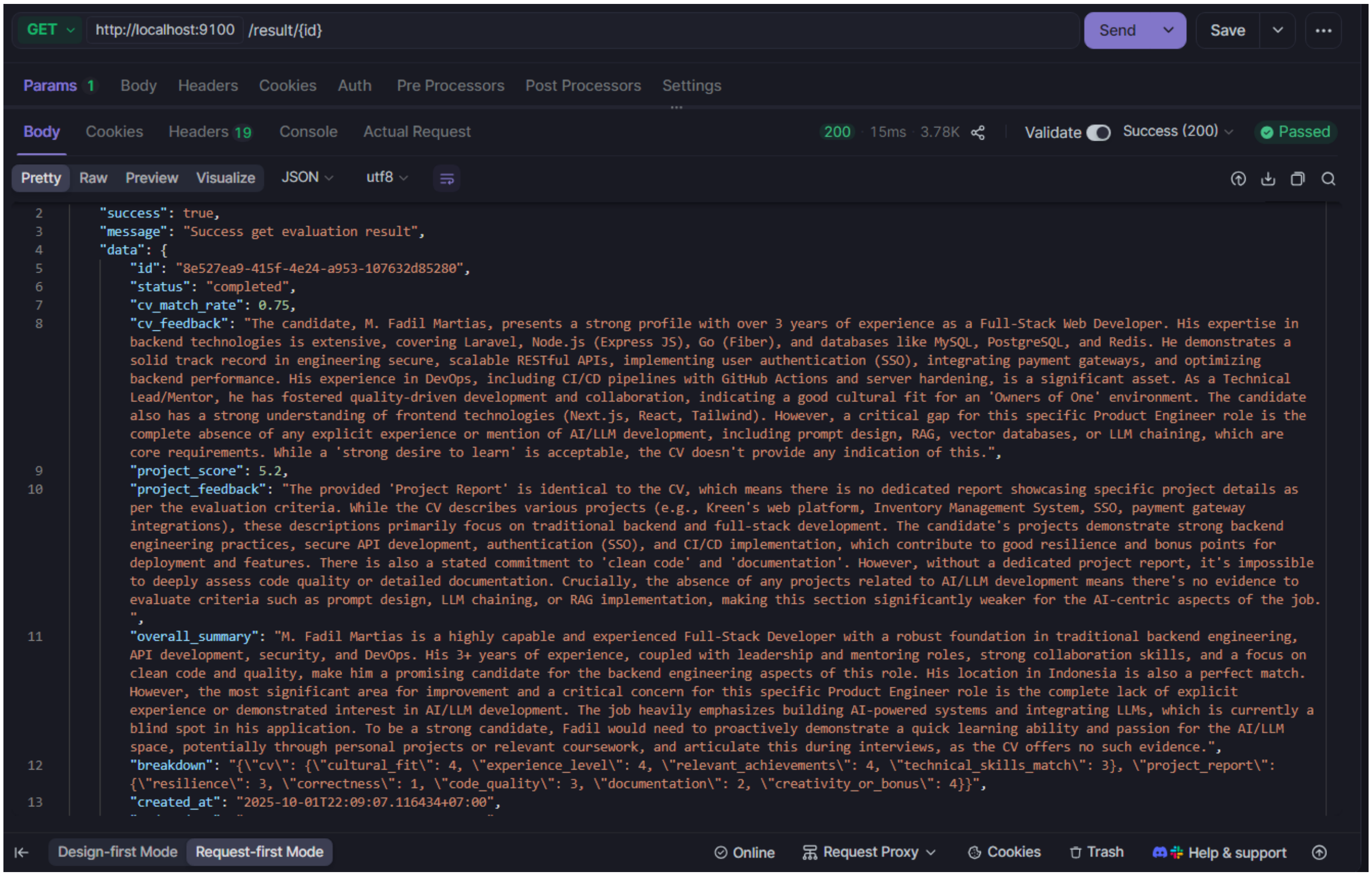
---

6. Screenshots of Real Responses

- GET /evaluate



POST /result/{id}



7. (Optional) Bonus Work

I added extra feature that can show breakdown of scoring metrics for cv and project_report