Study Case Submission

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

---

1. Title: CV EVALUATOR NESTJS

---

2. Candidate Information

- **Full Name: Fido Jahfal Prayoga**
- **Email Address: fidojahfalprayoga@gmail.com**

---

3. Repository Link

- github.com/fidojahfal/cv-evaluator-nestjs

---

4. Approach & Design (Main Section)

- **Initial Plan**

  I started the plan by dividing the problem into three main components:

  1. File Intake Layer

     Manages the candidate document upload process (CV and Project Report) using Multer and stores the metadata in a PostgreSQL database.
  2. Evaluation Pipeline

     The evaluation process is performed asynchronously using BullMQ and Redis, so the API does not have to wait for time-consuming AI processes. Workers read PDF files from local storage, extract text using pdf-parse, then process it using LangChain + HuggingFace API to generate automatic evaluations.
  3. Ground Truth Reference

     The system uses retrieval-based evaluation with ChromaDB as a vector database, which contains reference documents such as Job Descriptions, Case Study Briefs, and Scoring Rubrics. The embedding results from these documents are used as a reference for comparison with the contents of CVs and Project Reports.

- **System & Database Design**
  - API Endpoint Design

Endpoint	Method	Description
/upload	POST	This API used for uploading CV & Project Report as pdf.
/evaluate	POST	This API used for enqueue evaluation job with BullMQ + Langchain + HuggingFace.
/result/:id	GET	This API used for get the result of evaluation, if not finished no result returned.

  - Database schema.

    1. Schema File

       Storing information about files uploaded by candidates.

       Column:

       - id: unique file id
       - name: filename of the user file
       - path: storage location in the system
       - type: the type of file (cv or project_report)

    2. Schema Evaluation

       Storing assessment results from AI.

       Column:

       - id: unique evaluation id
       - status: evaluation process status (queued, processing, complete, failed)
       - result: result in JSON format (cv_match_rate, project_score, feedback and summary)
  - Job queue / long-running task handling.

    A. The AI evaluation process takes a long time because it involves:

       1. Text extraction from PDF.
       2. Embedding and retrieval from ChromaDB.
       3. Calls to LLM models (HuggingFace).

       To keep the main API responsive, this system uses BullMQ (based on Redis) to process tasks asynchronously.

B. Workflow:

1. The client sends an evaluation request to /evaluate.
2. The evaluate service adds the job to the evaluation queue.
3. The worker (evaluate.processor.ts) retrieves the job from the queue and executes:
   - Parsing the PDF file.
   - Evaluation with LangChain + HuggingFace.
   - Storing the results in PostgreSQL.
4. Once completed, the job status is updated to completed.

C. Advantages:

1. The API does not need to wait for the AI process (non-blocking).
2. Add more workers for parallel jobs.
3. If an error occurs, BullMQ automatically retries.
4. Evaluation results can be retrieved at any time via /result/:id endpoint.

- **LLM Integration**
  - Why you chose a specific LLM or provider.
    1. Using HuggingFace provider (via langchain)
    2. Using HuggingFace model.

    Selected for its high performance, speed, and relatively low cost.
  - Prompt design decisions.

    The system uses three separate prompt stages:
    1. Compare CVs with Job Descriptions and Scoring Rubrics.
    2. Compare the Project Report with the Case Study Brief and Project Rubric.
    3. Drawing a final conclusion based on the results of the previous two stages.
  - RAG (retrieval, embeddings, vector DB) strategy.
    1. Reference documents (job_description.pdf, case_study_brief.pdf, scoring_rubric.pdf) are uploaded to ChromaDB.
    2. During evaluation, the system retrieves the most relevant embeddings to add context to the prompt (retrieval augmented).

- **Prompting Strategy**

1. Cv Prompt

```
   const cvPrompt = `You are an AI evaluator comparing a candidate's CV against a job
description and a scoring rubric.

   --- Job Description ---
   ${jobDescriptionText.slice(0, 4000)}

   --- CV Scoring Rubric ---
   ${scoringRubricText.slice(0, 4000)}

   --- Candidate CV ---
   ${cvText.slice(0, 4000)}

   Respond ONLY in strict JSON format:
   {
     "cv_match_rate": number (0.0 - 1.0),
     "cv_feedback": string
   }`;
```

2. Project Prompt

```
3. const projectPrompt = `You are an AI evaluator reviewing a candidate's project
   report relative to the official case study and scoring rubric.
4.
5. --- Case Study Brief ---
6. ${caseStudyBriefText.slice(0, 4000)}
7.
8. --- Project Scoring Rubric ---
9. ${scoringRubricText.slice(0, 4000)}
10.
11.--- Candidate Project Report ---
```

```
${projectText.slice(0, 4000)}

Respond ONLY in strict JSON format:
{
  "project_score": number (0.0 - 5.0),
  "project_feedback": string
}
  `;
```

- **Resilience & Error Handling**

  I use error handling in my project as follows:

  1. All AI processes are run on BullMQ Worker, so the main API is not blocked.
  2. If HuggingFace fails to respond, the job will be automatically retried by BullMQ.
  3. Parsing of AI results JSON is wrapped in try/catch to prevent crashes.
  4. A default fallback is returned if the model does not return valid JSON.
  5. All PDF files are verified for size and MIME type upon upload.

- **Edge Cases Considered**

  1. Uploading files other than PDF automatically rejected.
  2. Empty or corrupt files will be skipped with an error message.
  3. HuggingFace timeout job automatically retries.
  4. Double evaluation of the same user prevented by unique ID checking.

5. Results & Reflection

- **Outcome**

1. The system successfully performs automatic uploading, enqueuing, and evaluation.
2. The JSON output is consistent and can be read directly by the frontend.
3. LLM provides relevant results based on the contents of CVs and Project Reports.

- **Evaluation of Results**

  1. The evaluation score is fairly stable (±0.05 deviation between tests).
  2. The variation in results appears due to LLM randomness (temperature = 0.2).
  3. The average results show a logical evaluation consistent with the document content.

- **Future Improvements**

  1. Adding caching for ground truth embedding results.
  2. Replacing pdf-parse with a PyMuPDF-based parser (more accurate).
  3. Using LangGraph to make the AI pipeline more modular.
  4. Adding Bull Board UI to monitor the job queue in real time.

6. Screenshots of Real Responses


```
{
    "cv": {
        "id": 7,
        "name": "1759940675889-720558134-CV_Fido Jahfal Prayoga_2025.pdf"
    },
    "project_report": {
        "id": 8,
        "name": "1759940675891-179043711-Study Case Submission.pdf"
    }
}
```

```
{
    "id": 38,
    "status": "completed",
    "result": {
        "cv_feedback": "The candidate has a strong technical background and relevant experience in web
            development. Their projects demonstrate a good understanding of various technologies and
            methodologies. However, the CV lacks specific details about achievements and quantifiable
            results.  Adding concrete examples of impact would strengthen the candidate's profile.",
        "cv_match_rate": 0.75,
        "project_score": 3.5,
        "overall_summary": "The candidate possesses a solid technical foundation and relevant
            experience, as evidenced by their CV and project. Their project showcases a good
            understanding of backend concepts and technologies.  However, both the CV and project
            could benefit from more quantifiable achievements and a stronger emphasis on documentation
            and testing.",
        "project_feedback": "The candidate demonstrates a good understanding of the project
            requirements and has implemented a well-structured system. The use of asynchronous
            processing and a vector database for retrieval-based evaluation are strong points.
            However, the project could benefit from more detailed documentation and a thorough testing
            strategy."
    }
}
```