1. Title

CANDIDATE EVALUATION SYSTEM

2. Candidate Information

Full Name:
Rafael A. C. Manurung

Email Address:
rafaelmanurung80@gmail.com

3. Repository Link

Link:
sdfasdfasdf.github

4. Approach & Design (Main Section)

a. Initial Plan

The main objective of this project is to build a Fast and Reliable Candidate Evaluation Pipeline System by automating the assessment process of Curriculum Vitae (CV) and technical Project Reports. This system is designed to reduce the burden of manual screening, accelerate the recruitment process, and provide objective and structured feedback on the candidate's suitability for the predefined job requirements (Job Role).

Technology and Initial Architecture
The initial approach of this project focuses on building a functional Minimum Viable Product (MVP) while leveraging the advantages of open-source ecosystems and easily accessible AI services:

- Backend Core: The application is developed using Node.js and Express to build RESTful APIs. This choice is based on Node.js's non-blocking nature and extensive library availability, enabling the development of responsive endpoints.
- Relational Database: MySQL is used as the primary database to store evaluation metadata, job statuses, and job context information. Prisma is used as an Object-Relational Mapping (ORM) tool to manage database schemas and interactions, ensuring data security and consistency.
- LLM Model: Gemini 2.5 Flash is chosen as the main Large Language Model. This choice is prioritized due to free access (within the developer tier) and an optimal balance between inference speed and sufficient reasoning capabilities for text analysis tasks.
- RAG Vector Database: For the implementation of Retrieval-Augmented Generation (RAG), ChromaDB is used as the Vector Database. ChromaDB serves as a knowledge store to store chunks of reference documents (such as detailed job descriptions or other RAG context documents), which can then be semantically retrieved to enhance LLM prompt accuracy and context.

b. System & Database Design

System Architecture and Service Segmentation

The system is designed based on the principle of Service Modularity, although executed within a single repository, with the goal of being easily scalable into microservices in the future. The main logic is divided as follows:

- Controller & Routes: Responsible for handling HTTP requests and responses, validating inputs, and initiating the evaluation process.
- Job Service (Pipeline Core): Serves as the orchestration service (main coordinator). This service manages the complete workflow—from data retrieval, text extraction, RAG Service orchestration, LLM Service invocation, to storing the final results.
- LLM Service: Responsible for interacting with the Gemini 2.5 Flash API. Its tasks include receiving content (CVs/Reports) and context-enriched prompts, and ensuring the output is in structured JSON format.
- RAG Service: The service that interacts with ChromaDB. Its function is to receive queries (e.g., job titles or specific keywords) and return the most relevant text context to enrich the LLM prompt.

Relational Database Design (MySQL via Prisma)

MySQL Database is used to maintain data integrity and relationships between the documents being evaluated and their evaluation results. Two main entities are implemented:

Table	Main Purpose	Description
Document	Store metadata and physical file paths of uploaded CVs or Project Report	Enables the system to locate and extract text from files when the evaluation process begin
Evaluation	Store status (processing, completed, failed, queue) and JSON evaluation return by LLM	Acts as both job queue and a final results repository. Has a relation Document entity

Workflow (Data Flow)

The evaluation workflow is a synchronous process triggered by the Controller and orchestrated by the Job Service:

- Initiation: Users upload CV and Project Report files. The system stores the file paths in the Document table and creates a new entry in the Evaluation table with a processing status.
- RAG Preparation: The Job Service calls the RAG Service to prepare context. During initial setup, a separate script has created two specific collections in ChromaDB: cv_context (for Job Descriptions and hard skill criteria) and project_context (for project-specific technical criteria).
- LLM Orchestration: The Job Service retrieves relevant context from the RAG Service (using the job role as a query) and combines it with the extracted content from the CV/Report.
- Evaluation Invocation: The Job Service calls the LLM Service once with all content (CV, Report, CV Context, Project Context) and requests a unified JSON output.
- Completion: The structured JSON result (including match rate, scores, and summary) is stored in the result column of the Evaluation table, and the status is updated to completed.

c. LLM Integration

The integration of a Large Language Model (LLM) is the core of this evaluation system. Selecting the right model and implementing proper prompting techniques is crucial to ensure accurate and structured outputs.

Model Selection: Gemini 2.5 Flash

Gemini 2.5 Flash was chosen as the primary evaluation engine based on the following strategic considerations:

- Accessibility & Rapid Development: This model is easily accessible through Google AI Studio using only an API Key, accelerating development and prototyping.
- Cost Efficiency: A free tier is available, which is friendly for project development and skill development, allowing extensive testing without cost barriers.
- Performance Balance: Gemini 2.5 Flash offers an optimal balance between inference speed (important for responsive APIs) and strong reasoning capabilities, necessary for analyzing long text documents such as CVs and technical reports.
- Monitoring: Google AI Studio provides a dashboard that facilitates API request monitoring, helping track usage and enabling efficient debugging.

Ensuring Structured Output

To guarantee that the LLM returns evaluation results in a specific JSON format (e.g., cv_match_rate, project_score, etc.) as required by the pipeline, the Structured Output feature of the Gemini SDK is used.

Implemented techniques include:

- Response Schema: A strict JSON schema is defined in code, outlining each field, its data type (Type.NUMBER, Type.STRING), and description.
- MIME Type Enforcing: By setting responseMimeType: "application/json" in the API call, Gemini is explicitly instructed to produce valid JSON output that adheres to the given schema, eliminating the need for error-prone JSON string parsing or cleaning on the backend.

Retrieval-Augmented Generation (RAG) Integration

To improve accuracy and context, RAG integration is implemented using ChromaDB.

- ChromaDB as Vector Database: ChromaDB stores two context collections: cv_context (for job descriptions and mandatory hard skill criteria) and project_context (for project-specific technical specifications or criteria).
- Process: Before invoking Gemini, the RAG Service retrieves the most relevant context from ChromaDB (based on job titles or candidate keywords) and injects this context directly into the LLM prompt. This allows Gemini to perform evaluations based on the most up-to-date and specific reference criteria.

d. Prompting Strategy

The prompting strategy is designed to guide Gemini to act as an expert evaluator and ensure the model uses all provided data to generate structured outputs. This strategy consists of three main components:

Persona and Objective Setting (System Instruction)

- Dual Persona: The AI model is defined as an “AI Recruitment Expert” with the capability to perform in-depth technical document analysis (CVs) and assess engineering quality (Project Reports). This persona encourages a more formal, analytical, and objective response style.
- Single Objective: The primary instruction asks Gemini to perform a comprehensive evaluation based on all input context and explicitly instructs it to produce output that conforms to the predefined JSON schema.

Emphasis on Detailed Analysis and Rubrics

- Detailed Analysis: The model is instructed to conduct a "detailed analysis" and provide feedback aligned with implicit evaluation criteria (such as scoring rubrics).
- Quality Assurance: Even though the scoring rubrics are not explicitly injected into each field (to keep the prompt concise), the model is instructed to ensure that feedback and scores (cv_match_rate and project_score) logically correlate with the analysis it produces.

Unified Output

- By performing only a single LLM call, the model is forced to synthesize the results from both the CV and Project, leading to a more comprehensive and integrated overall summary (overall_summary).

e. Resiliance & Error Handling

Although the evaluation pipeline is implemented synchronously (within the API request-response cycle), system resilience is a top priority, especially due to dependencies on external services (LLM API and databases). The error handling strategy focuses on isolating failures, recovering from external issues, and ensuring data quality.

Major Failure Handling (Try-Catch)

- Error Isolation: Every critical function in the pipeline, such as text extraction, RAG context retrieval, and LLM API invocation, is wrapped in a try...catch block. This ensures that a failure at one stage (e.g., PDF parsing failure) does not crash the entire application.
- Status Logging: When an error occurs (within the catch block), the status in the Evaluation table is immediately updated to failed, along with detailed error log messages. This allows developers to quickly identify and diagnose the source of the problem without needing to sift through extensive server logs.

f. Edge Cases Considered

In addition to external API failures, several edge cases have been explicitly considered and handled within the workflow to ensure data quality and prevent pipeline failures:

Empty Input Documents or Extraction Failure

Handling: The Job Service immediately checks the length of the cvContent and reportContent strings after extraction. If either value is zero (0), the process is halted. The Evaluation status is set to failed with a clear error message, such as:
"Failed to extract text. Document is empty or unreadable."

RAG Context Irrelevant or Unavailable

Prompt Instruction: The system instruction in the LLM prompt includes a directive to evaluate based on general knowledge and the content of the provided document if the injected context is insufficient. This ensures the evaluation continues, although the accuracy may be slightly compromised.

LLM Token Limit Exceeded (Context Window Limit)

Handling: If the LLM cannot process due to exceeding the token limit, the service is unable to proceed and must handle this scenario as a failure (e.g., logging the issue and updating the Evaluation status accordingly).

5. Results & Reflection

a. Outcome

What Worked Well:

The evaluation pipeline is fast and reliable, completing end-to-end processing—including text extraction, RAG context retrieval, and LLM (Gemini 2.5 Flash) evaluation—in under 60 seconds per candidate, while consistently producing structured JSON output enhanced by RAG-injected context.

What Didn't Work Well:

Performance issues arose with very large documents due to the synchronous pipeline, and occasional LLM API unavailability (503 errors) caused total pipeline failures, highlighting limitations in scalability and maintainability.

b. Evaluation of Results

The output quality of the LLM (Gemini 2.5 Flash) is generally considered stable and high-quality:

- Stability: Consistency is mainly achieved through a strict response schema that enforces uniform JSON output and RAG context injection, which makes feedback specific, consistent, and objective.
- Limitations: Minor inconsistencies can occur in evaluating non-technical experience (e.g., very short tenure in CVs), as the LLM tends to focus more on hard skills provided by RAG and may overlook significant differences in work duration.

c. Future Improvements

Database Structure and Entities Refactoring:

Implement separate User and Job entities to better track evaluation history per user and facilitate document retrieval. Redesign the overall database structure to support faster indexing and queries.

LLM Quality and Reliability Enhancement:

Switch to a more robust LLM (e.g., Gemini 2.5 Pro) for critical evaluations, particularly for narrative analysis and complex reasoning. Implement fine-tuning on smaller models for specific tasks (e.g., CV data extraction) to improve accuracy, reduce latency, and lower costs.

User Interface Development:

Build a front-end interface that allows recruiters to upload files, monitor pipeline status in real-time, and view structured evaluation results through an interactive dashboard.

6. Screenshots of Real Response

asdfadsf

7. (Optional) Bonus Work