
TalentSieve RAG Talent Screener

---

1. Title

Project: TalentSieve RAG Talent Screener — Automated CV & Project Evaluation Backend

---

2. Candidate Information

- **Full Name:** Muhammad Ulil 'Azmi
- **Email Address:** *ulilazmi100@gmail.com*

---

3. Repository Link

GitHub Repository: https://github.com/ulilazmi100/TalentSieve-talent-screener

⚠ Repository information:
The repository includes a reproducible DEMO_MODE, full Docker setup, and Jest test coverage.

---

4. Approach & Design

Initial Plan

Goal: build a backend API that automates evaluation of candidate CVs and project reports using an LLM (Gemini) and RAG pipeline.

Breakdown of requirements

1. Accept and store CV + project report uploads.
2. Create a background evaluation job (non-blocking).
3. Evaluate via RAG + LLM scoring.
4. Return a structured JSON result with 5 fields (cv_match_rate, cv_feedback, project_score, project_feedback, overall_summary).

Key assumptions & scope boundaries

- Offline demo mode (DEMO_MODE=true) must be self-contained and reproducible.

- Evaluation should always produce a complete JSON result, even if the LLM fails.
- Real LLM and Qdrant integrations optional for reviewers.

System & Database Design

API Design

Endpoint	Method	Description	Example Response
/upload	POST (multipart)	Upload CV and project PDFs	{ "cv_id": "file_xxx", "project_id": "file_xxx" }
/evaluate	POST (JSON)	Enqueue evaluation job	{ "id": "job_xxx", "status": "queued" }
/result/{id}	GET	Retrieve job status and result	{ "id": "job_xxx", "status": "completed", "result": {...} }

Database Schema

- **documents** — stores file metadata
  id, filename, type, storage_path, extracted_text, created_at
- **jobs** — tracks each evaluation job
  id, job_title, cv_id, project_id, status, result (JSONB), worker_logs, timestamps
- **demo_db.json** — local file-based fallback database when DEMO_MODE=true.

Job Queue

- Implemented using **BullMQ + Redis**.
- Jobs are queued by the API server and processed by a **worker** (src/worker.js).

LLM Integration

Chosen Provider: Google Gemini (via src/lib/aiClient.js)
Why: simple REST integration, cost-efficient, easy to replace.

Fallback: deterministic heuristic scoring when LLM unavailable (in demo mode).
Temperature: fixed at 0.0 for reproducibility.

Prompt Design & RAG Strategy

RAG:

- Extracts text from PDFs (pdf-parse)
- Chunks text (1,200 chars with 200 overlap)
- Embeds using Gemini (or random vector in demo)
- Retrieves context from Qdrant (when live)

Prompting (real snippets):

// CV Evaluation Prompt

Evaluate the candidate's CV for job title: {jobTitle}.

Return JSON with integer scores (1–5) for technical_skills, experience_level,

relevant_achievements, cultural_fit, and short cv_feedback (1–3 sentences).

// Project Evaluation Prompt

Evaluate the project report for job title: {jobTitle}.

Return JSON with integer scores (1–5) for correctness, code_quality,

resilience, documentation, creativity, and project_feedback (2–4 sentences).

Resilience & Error Handling

- **AJV validation** ensures JSON format correctness.
- **Fallback scorer** (src/fallback/scorer.js) provides deterministic numeric values.
- **Retries/backoff** for Qdrant & LLM calls.
- **Graceful shutdown** (app.shutdown()) ensures tests terminate cleanly.

Edge Cases Considered

- Missing/invalid PDF → fallback to raw text extraction.
- LLM returns malformed JSON → auto-repair & fallback scoring.
- External services down → demo mode continues offline.
- Empty content → produces neutral scoring with zero errors.

---

5. Results & Reflection

Outcome

✅ What worked well

- End-to-end RAG pipeline runs deterministically in demo mode.
- All five required fields generated correctly.
- Integration tests (Jest) pass consistently.
- Docker infra for Redis/Postgres/Qdrant provided and reproducible.

⚠ What didn’t work / limitations

- Live Gemini calls require API key (not executed in CI).
- No authentication or rate-limiting (for demo simplicity).
- Uploaded PDFs stored in plaintext (for local testing).

Evaluation of Results

Real test evidence (from your terminal):

```
$ curl http://localhost:3000/result/job_0c02f13a-0331-4980-9800-dcd9ed376327 | jq
{
 "id": "job_0c02f13a-0331-4980-9800-dcd9ed376327",
 "status": "completed",
 "result": {
  "cv_match_rate": 0.73,
  "cv_feedback": "Strong backend tech footprint. Has measurable achievements. Mentions collaboration/culture keywords.",
  "project_score": 3.2,
  "project_feedback": "Includes testing or validation mentions. Contains documentation cues. Shows creative elements",
  "overall_summary": "Good candidate fit with some areas to improve; consider for interview with targeted questions. CV note: Strong backend tech footprint. Project note: Includes testing or validation mentions. Contains documentation cues."
 }
}
```

All five canonical fields are present and correctly validated ✅

Future Improvements

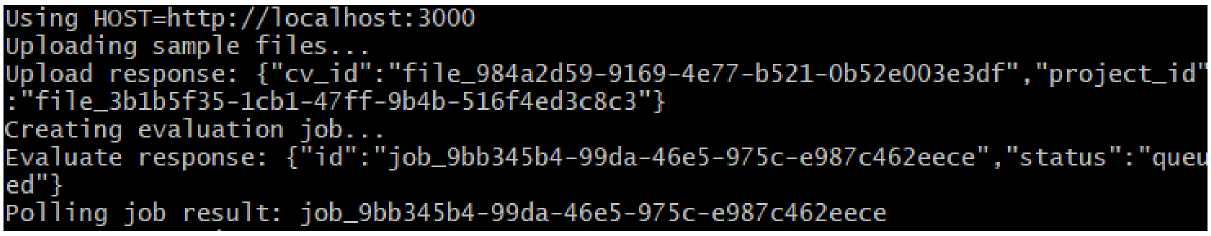
- Add JWT/API key authentication.
- Encrypt file storage and add antivirus scanning.
- Add Prometheus metrics + Grafana dashboards.
- Expand fallback heuristics with lightweight ML model.
- Add CI workflow running Postgres + Qdrant integration tests.

---

6. Screenshots of Real Responses

Include screenshots of:

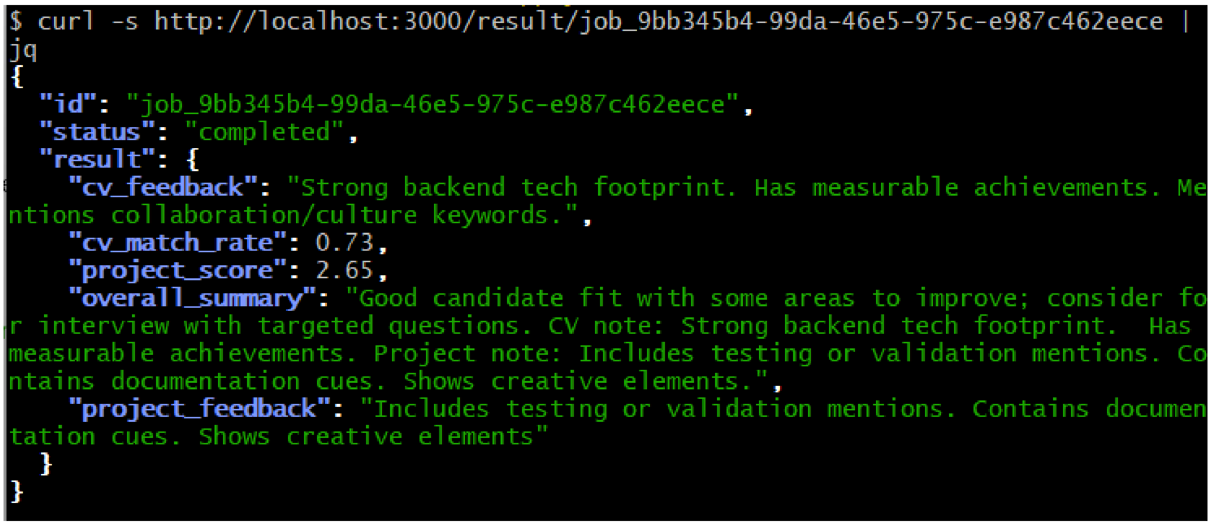
1. Upload response → job_id + status



```
Using HOST=http://localhost:3000
Uploading sample files...
Upload response: {"cv_id":"file_984a2d59-9169-4e77-b521-0b52e003e3df","project_id"
:"file_3b1b5f35-1cb1-47ff-9b4b-516f4ed3c8c3"}
Creating evaluation job...
Evaluate response: {"id":"job_9bb345b4-99da-46e5-975c-e987c462eece","status":"queu
ed"}
Polling job result: job_9bb345b4-99da-46e5-975c-e987c462eece
```

```
$ HOST=http://localhost:3000 bash scripts/run_sample_job.sh
Using HOST=http://localhost:3000
Uploading sample files...
Upload response: {"cv_id":"file_984a2d59-9169-4e77-b521-
0b52e003e3df","project_id":"file_3b1b5f35-1cb1-47ff-9b4b-516f4ed3c8c3"}
Creating evaluation job...
Evaluate response: {"id":"job_9bb345b4-99da-46e5-975c-
e987c462eece","status":"queued"}
Polling job result: job_9bb345b4-99da-46e5-975c-e987c462eece
```

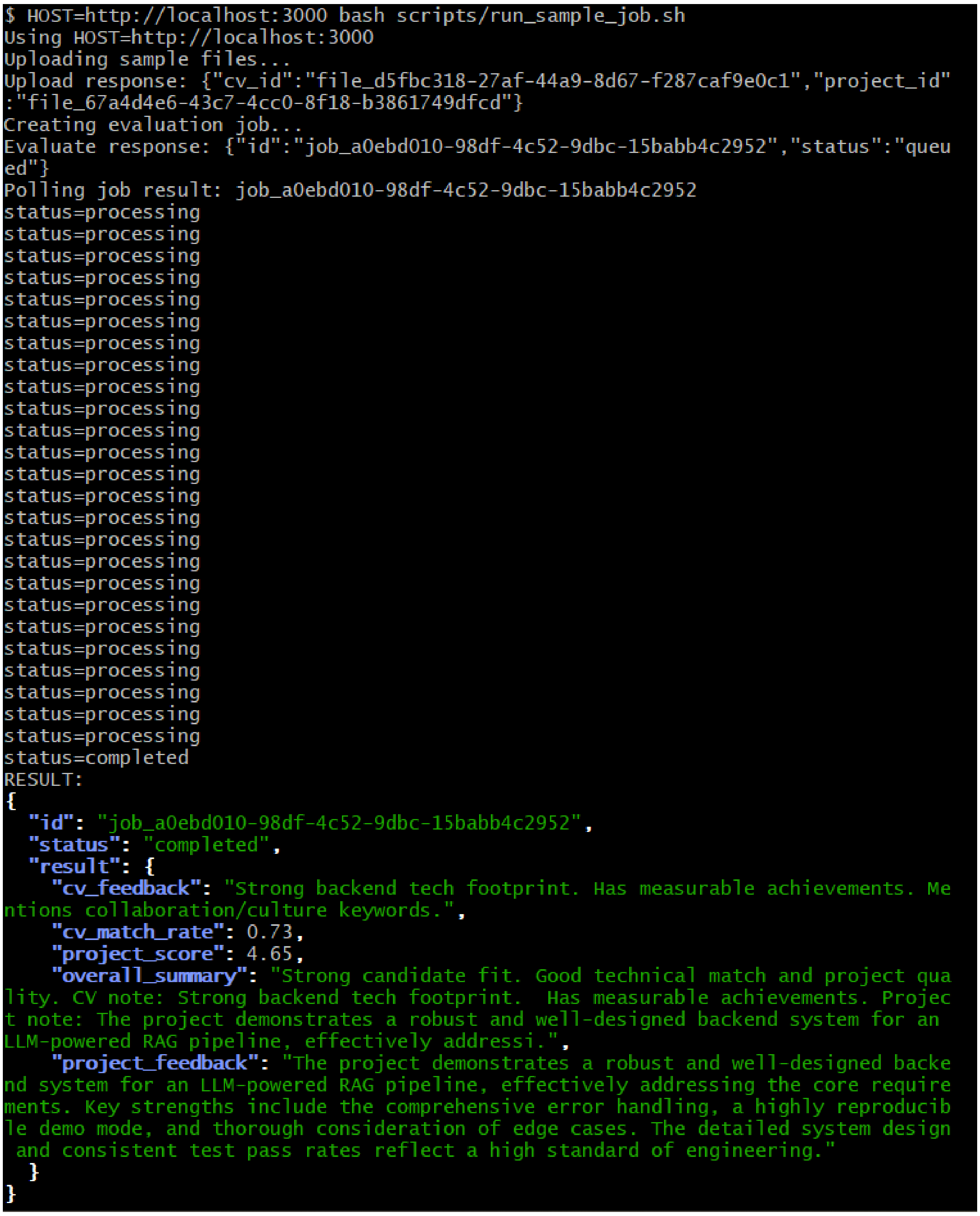
2. The final JSON result as above.



```
$ curl -s http://localhost:3000/result/job_9bb345b4-99da-46e5-975c-e987c462eece |
jq
{
  "id": "job_9bb345b4-99da-46e5-975c-e987c462eece",
  "status": "completed",
  "result": {
    "cv_feedback": "Strong backend tech footprint. Has measurable achievements. Me
ntions collaboration/culture keywords.",
    "cv_match_rate": 0.73,
    "project_score": 2.65,
    "overall_summary": "Good candidate fit with some areas to improve; consider fo
r interview with targeted questions. CV note: Strong backend tech footprint.  Has
measurable achievements. Project note: Includes testing or validation mentions. Co
ntains documentation cues. Shows creative elements.",
    "project_feedback": "Includes testing or validation mentions. Contains documen
tation cues. Shows creative elements"
  }
}
```

```
$ curl -s http://localhost:3000/result/job_9bb345b4-99da-46e5-975c-e987c462eece |
jq
{
  "id": "job_9bb345b4-99da-46e5-975c-e987c462eece",
  "status": "completed",
  "result": {
    "cv_feedback": "Strong backend tech footprint. Has measurable achievements.
Mentions collaboration/culture keywords.",
    "cv_match_rate": 0.73,
    "project_score": 2.65,
    "overall_summary": "Good candidate fit with some areas to improve; consider for
interview with targeted questions. CV note: Strong backend tech footprint.  Has
measurable achievements. Project note: Includes testing or validation mentions.
Contains documentation cues. Shows creative elements.",
    "project_feedback": "Includes testing or validation mentions. Contains
documentation cues. Shows creative elements"
  }
}
```

3. Full run sample result:

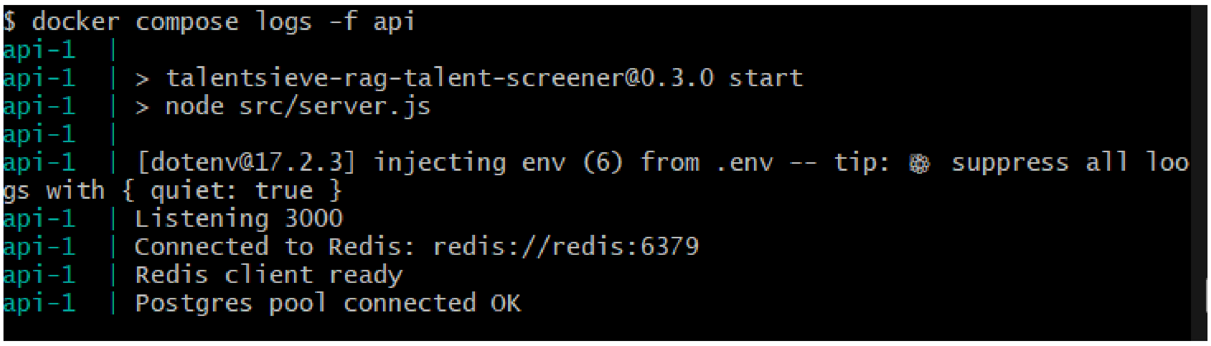


```
$ HOST=http://localhost:3000 bash scripts/run_sample_job.sh
Using HOST=http://localhost:3000
Uploading sample files...
Upload response: {"cv_id":"file_d5fbc318-27af-44a9-8d67-f287caf9e0c1","project_id"
:"file_67a4d4e6-43c7-4cc0-8f18-b3861749dfcd"}
Creating evaluation job...
Evaluate response: {"id":"job_a0ebd010-98df-4c52-9dbc-15babb4c2952","status":"queu
ed"}
Polling job result: job_a0ebd010-98df-4c52-9dbc-15babb4c2952
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=completed
RESULT:
{
  "id": "job_a0ebd010-98df-4c52-9dbc-15babb4c2952",
  "status": "completed",
  "result": {
    "cv_feedback": "Strong backend tech footprint. Has measurable achievements. Me
ntions collaboration/culture keywords.",
    "cv_match_rate": 0.73,
    "project_score": 4.65,
    "overall_summary": "Strong candidate fit. Good technical match and project qua
lity. CV note: Strong backend tech footprint.  Has measurable achievements. Projec
t note: The project demonstrates a robust and well-designed backend system for an
LLM-powered RAG pipeline, effectively addressi.",
    "project_feedback": "The project demonstrates a robust and well-designed backe
nd system for an LLM-powered RAG pipeline, effectively addressing the core require
ments. Key strengths include the comprehensive error handling, a highly reproducib
le demo mode, and thorough consideration of edge cases. The detailed system design
 and consistent test pass rates reflect a high standard of engineering."
  }
}
```

```
$ HOST=http://localhost:3000 bash scripts/run_sample_job.sh
Using HOST=http://localhost:3000
Uploading sample files...
Upload response: {"cv_id":"file_d5fbc318-27af-44a9-8d67-
f287caf9e0c1","project_id":"file_67a4d4e6-43c7-4cc0-8f18-b3861749dfcd"}
Creating evaluation job...
Evaluate response: {"id":"job_a0ebd010-98df-4c52-9dbc-
15babb4c2952","status":"queued"}
Polling job result: job_a0ebd010-98df-4c52-9dbc-15babb4c2952
status=processing
status=processing
```

```
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=processing
status=completed
RESULT:
{
  "id": "job_a0ebd010-98df-4c52-9dbc-15babb4c2952",
  "status": "completed",
  "result": {
    "cv_feedback": "Strong backend tech footprint. Has measurable achievements.
Mentions collaboration/culture keywords.",
    "cv_match_rate": 0.73,
    "project_score": 4.65,
    "overall_summary": "Strong candidate fit. Good technical match and project
quality. CV note: Strong backend tech footprint.  Has measurable achievements.
Project note: The project demonstrates a robust and well-designed backend system
for an LLM-powered RAG pipeline, effectively addressi.",
    "project_feedback": "The project demonstrates a robust and well-designed
backend system for an LLM-powered RAG pipeline, effectively addressing the core
requirements. Key strengths include the comprehensive error handling, a highly
reproducible demo mode, and thorough consideration of edge cases. The detailed
system design and consistent test pass rates reflect a high standard of
engineering."
  }
}
```

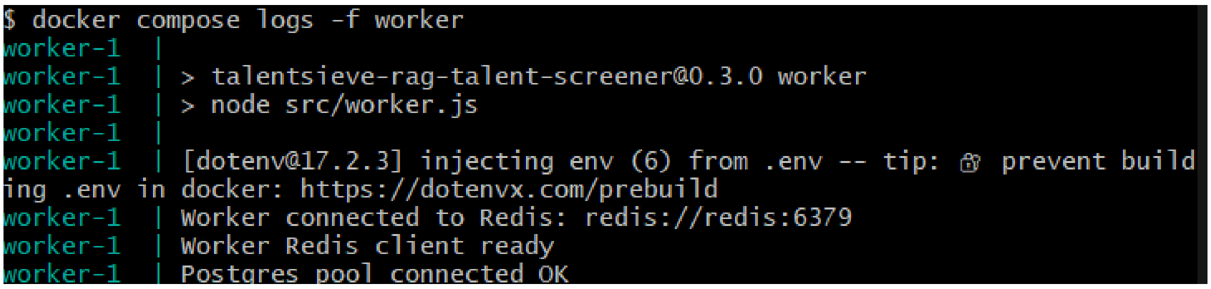
4. Docker Compose logs for API:



```
$ docker compose logs -f api
api-1  |
api-1  | > talentsieve-rag-talent-screener@0.3.0 start
api-1  | > node src/server.js
api-1  |
api-1  | [dotenv@17.2.3] injecting env (6) from .env -- tip: ⚙ suppress all loo
gs with { quiet: true }
api-1  | Listening 3000
api-1  | Connected to Redis: redis://redis:6379
api-1  | Redis client ready
api-1  | Postgres pool connected OK
```

```
$ docker compose logs -f api
api-1  |
api-1  | > talentsieve-rag-talent-screener@0.3.0 start
api-1  | > node src/server.js
api-1  |
api-1  | [dotenv@17.2.3] injecting env (6) from .env -- tip: ⚙  suppress all
loogs with { quiet: true }
api-1  | Listening 3000
api-1  | Connected to Redis: redis://redis:6379
api-1  | Redis client ready
api-1  | Postgres pool connected OK
```

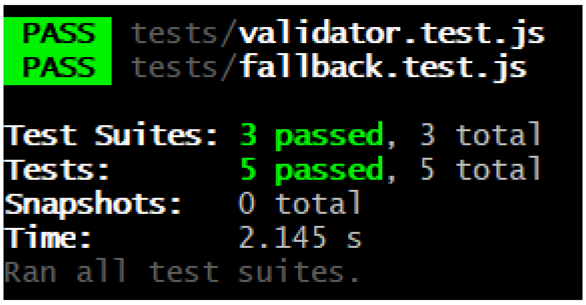
5. Docker Compose logs for Worker



```
$ docker compose logs -f worker
worker-1  |
worker-1  | > talentsieve-rag-talent-screener@0.3.0 worker
worker-1  | > node src/worker.js
worker-1  |
worker-1  | [dotenv@17.2.3] injecting env (6) from .env -- tip: 🔐 prevent build
ing .env in docker: https://dotenvx.com/prebuild
worker-1  | Worker connected to Redis: redis://redis:6379
worker-1  | Worker Redis client ready
worker-1  | Postgres pool connected OK
```

```
$ docker compose logs -f worker
worker-1  |
worker-1  | > talentsieve-rag-talent-screener@0.3.0 worker
worker-1  | > node src/worker.js
worker-1  |
worker-1  | [dotenv@17.2.3] injecting env (6) from .env -- tip: 🔐 prevent
building .env in docker: https://dotenvx.com/prebuild
worker-1  | Worker connected to Redis: redis://redis:6379
worker-1  | Worker Redis client ready
worker-1  | Postgres pool connected OK
```

6. Jest test summary:



```
 PASS  tests/validator.test.js
 PASS  tests/fallback.test.js

Test Suites: 3 passed, 3 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.145 s
Ran all test suites.
```

```
PASS  tests/validator.test.js
 PASS  tests/fallback.test.js

Test Suites: 3 passed, 3 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        2.145 s
Ran all test suites.
```

7. (Optional) Bonus Work

All bonus features are factual and verifiable from the repository:

#	Bonus Feature	Description
1	**Offline Demo Mode (DEMO_MODE=true)**	Allows full functionality (upload, evaluate, result) with no external dependencies. Uses .demo_db.json and mock embeddings.
2	**Canonical Key Normalization**	Accepts flexible JSON keys (cvDocId, projectId, etc.) and normalizes them for robustness.
3	**AJV Validation + Fallback Logic**	Ensures valid structured results even when LLM output is invalid.
4	**Integration & Unit Tests**	Automated Jest tests for end-to-end evaluation flow.
5	**Graceful Shutdown API**	Enables clean termination of async jobs in tests (app.shutdown()).
6	**Dockerized Infra Stack**	Fully functional local environment: Redis, Postgres, Qdrant, API, Worker.
7	**Deterministic Heuristic Scorer**	Stable keyword-based fallback scoring algorithm (src/fallback/scorer.js).

End of Report
