SHL GenAI Assessment — RAG Recommender (Draft Report)

Author: Rajdeep Samanta
Date: (fill)

Page 1 — Problem, Data and Architecture

Problem statement

Build a simple Retrieval-Augmented-Generation (RAG) style recommender for SHL assessment catalog that: given a short query (e.g., "Suggest a test for logical reasoning"), returns the most relevant assessments from the catalog in JSON format for automated evaluation.

Dataset

The provided dataset (`Gen_AI Dataset.xlsx`) contains assessment products with fields such as Product ID, Title, Description, Skills, Tags and Duration. I converted the spreadsheet into a JSON catalog (`products.json`) to make lookups simple for the backend.

Data cleaning

- Replaced missing values with empty strings.
- Normalized skills/tags to lowercase lists and removed empty tokens.
- Kept duration as integer minutes when available.

Architecture

- Retrieval: simple in-memory keyword scoring (title matches, description matches, skills/tags matches). This is intentionally lightweight and reproducible.
- API: Node.js + Express exposes `/api/recommend?query=<text>` and `/api/products`.
- Frontend: minimal static HTML UI to test queries.

Design choices

- A small deterministic scoring function was chosen for reliability and speed (no external or paid APIs).
- The converter script (`convert_to_json.py`) reads the Excel and creates `products.json` which the server consumes.

Page 2 — Implementation, Evaluation, Future Work

Implementation details

- Backend: Express (ES modules), serving static `index.html` and the JSON API. The server loads `products.json` at startup.
- Converter: Python script using pandas + openpyxl to convert Excel to JSON.
- Frontend: Vanilla JS to call the API and render results.

Evaluation

- Manual spot-checks: queries such as "logical reasoning" map to Logical Reasoning Test (P1). The scoring function ranks title matches higher than description matches and counts skill/tag matches as additional signals.

Challenges

- The provided instructions expect deployment. I include clear deployment steps in the README to deploy on Render or Railway.

Future improvements

- Replace keyword scoring with TF-IDF and cosine similarity (scikit-learn or lunr.js) for better retrieval.
- Add embeddings + vector DB (e.g., FAISS, Weaviate) for semantic search.
- Add caching, pagination, and more robust query parsing (stemming, stopwords).

Links

- API: (add render URL here after deploy)
- Web UI: (add render URL)
- GitHub: (add repo link)

End of draft. Export to PDF using any Markdown -> PDF exporter (VS Code Print, Pandoc or by pasting into Google Docs and exporting PDF).