

# AI Engineer Intern Assignment: File-Based Multimodal RAG for Test-Case / Use-Case Generation

## Goal

Build a **file-based Retrieval-Augmented Generation (RAG)** application that can take **multiple input sources** (e.g., text notes, PDFs/docs, images, attachments) and answer user queries like:

“Create use cases for user signup”

The system should **retrieve relevant context from the provided sources** and generate **high-quality use cases / test cases** grounded in that context.

You'll be scored primarily on **RAG implementation quality, accuracy, safeguards, modular design, and code quality, (UI has very less weightage)**.

---

## What you must build

### 1) Ingestion (file-based knowledge base)

Support uploading / adding a folder of files such as:

- Text/Markdown
- PDF / DOCX (or at least PDF)
- Images (PNG/JPG) — should be usable as context (via OCR and/or vision)
- Links (Optional)

You must:

- Parse/extract content per file type (including images).
- Chunk content (with a reasonable strategy).
- Store indexed chunks for retrieval (vector DB, local embeddings store, hybrid search, etc.).

- Keep everything **file-based/local** (no mandatory external DB required).

## 2) Retrieval

Given a query, retrieve the most relevant evidence chunks:

- Prefer **hybrid retrieval** (vector + keyword) if you can.
- Provide configurable top\_k, thresholds, reranking (optional but valued).
- **citations** in output is optional

## 3) Generation: Use cases / Test cases

For queries like “Create use cases for user signup”, generate:

- A clean **JSON** structure (e.g., **Use Case + Preconditions + Steps + Expected Result + Negative/Boundary cases**).
- Output must be **grounded in retrieved context** (don’t invent product features).
- If info is insufficient, the system should:
  - Ask clarifying questions **or**
  - Explicitly state assumptions + missing info.
- Always proper structured JSON output should be present

## 4) Guards & Checks (high weight)

Add practical protections such as:

- Hallucination reduction: “only answer using retrieved context”
- Minimum evidence threshold (if retrieval confidence is low → ask questions)
- Prompt-injection resilience (ignore instructions inside documents that try to override behavior)

- Deduplication / chunk quality checks
- Basic evaluation hooks (even a small suite) is a plus

## 5) Observability & Debugging (recommended)

Provide at least:

- “Show retrieved chunks” / debug mode
  - Logs for ingestion + retrieval + generation
  - Simple metrics (latency, number of chunks, token usage if possible)
- 

## Tech constraints (flexible)

You may use **any**:

- Language (Python/TS/Go/Rust/etc.)
- LLM provider (OpenAI/Anthropic/Bedrock/local model)
- Embeddings approach
- Storage layer (FAISS/LanceDB/Chroma/SQLite+vectors/etc.)

Keep setup developer-friendly:

- Clear README
  - .env.example
  - Minimal one-command run (Docker optional)
- 

## Deliverables

## A) GitHub Repository

- Push the full project to a GitHub repo.
- Grant access to:
  - `santhosh@devassure.io`
  - `divya@devassure.io`

Repository must include:

- README.md (setup + run + architecture + examples + tools, IDEs used for development)
- Source code (modular, linted where applicable)
- Sample dataset folder (small but representative)
- Optional: docs/ with design notes and tradeoffs

## B) Walkthrough Video (mandatory)

Record a **walkthrough video** (voice explanation is required) that covers:

- Architecture overview (ingestion → retrieval → generation)
- Demo with at least 2 queries
- How multimodal files are handled (especially images)
- What safeguards/checks you added and why
- How to run it locally

You can upload the video as:

- Unlisted YouTube link, or
- Google Drive link, or

- Any accessible link

## C) Submission

Email the **GitHub repo link + video file/link** to:

**santhosh@devassure.io**

**Deadline:** *As provided in the hiring email / posting.*

(Submit before the deadline mentioned; late submissions may not be evaluated.)

---

## Scoring rubric (what we care about most)

### High weight

- Correct, grounded retrieval (relevant chunks)
- Output accuracy and structure
- Guardrails and evidence-based answering
- Code quality, modularization, clean abstractions
- Sensible library/framework choices

### Medium weight

- Multimodal handling quality (images/PDFs)
- Basic eval tests

### Low weight

- UI polish
  - Observability / debug tooling
-

# Required output format (for generated test cases)

At minimum, produce:

- **Use Case Title**
  - **Goal**
  - **Preconditions**
  - **Test Data** (if applicable)
  - **Steps**
  - **Expected Results**
  - **Negative cases**
  - **Boundary cases**
- 

## Examples (3)

**Below examples are samples. Actual results should be JSON output**

### Example 1: PRD + API spec + UI screenshot

#### Input files

- PRD\_Signup.md
- API\_Spec.yaml
- Signup\_UI.png

#### Query

“Create use cases for user signup”

#### Sample use cases (illustrative)

## **UC1 — Signup with valid email + password**

- **Preconditions:** User is logged out
- **Test data:** email = new\_user@example.com, password = StrongPass#123
- **Steps:**
  - Open the Signup page.
  - Enter a valid email.
  - Enter a valid password and confirm password (if available).
  - Click **Create Account**.
- **Expected results:**
  - Account is created successfully.
  - User sees “Verify your email” message OR is redirected per PRD.

## **UC2 — Reject duplicate email signup**

- **Preconditions:** An account exists for existing@example.com
- **Steps:**
  - Open Signup page.
  - Enter existing@example.com and a valid password.
  - Click **Create Account**.
- **Expected results:**
  - Error message shown (e.g., “Email already exists”).
  - No new account created.

## **UC3 — Password policy validation**

- **Steps:**

- Open Signup page.
- Enter a valid email.
- Enter a weak password (e.g., 12345).
- Click **Create Account**.

- **Expected results:**

- Inline validation message describing password rules.
  - Signup is blocked.
- 

## **Example 2: Auth flow PDF + error codes**

### **Input files**

- Auth\_Flow.pdf
- Error\_Codes.txt

### **Query**

“Generate negative and boundary test cases for signup and verification”

### **Sample use cases (illustrative)**

#### **UC1 — Email verification with invalid token**

- **Preconditions:** User has signed up and received a verification link
- **Steps:**
  - Replace the token in the verification URL with an invalid token.
  - Open the modified verification URL.

- **Expected results:**

- Verification fails with a clear error state.
- Users are prompted to request a new verification link (if supported).

## UC2 — Email verification token expired

- **Steps:**

- Sign up and obtain verification link/token.
- Wait until token expiry time (or simulate expiry if supported).
- Open verification link.

- **Expected results:**

- “Token expired” (or equivalent) message.
- CTA to resend verification link.

## UC3 — Boundary: max email length

- **Steps:**

- Open Signup page.
- Enter an email at maximum allowed length (from docs).
- Submit signup.

- **Expected results:**

- Accepted if within bounds; rejected with validation if over bounds.

---

## Example 3: Support tickets + email template + known issues

### Input files

- Support\_Tickets.csv
- Email\_Template.html
- Known\_Issues.md

## Query

“Create use cases for signup that prevent known issues from recurring”

### Sample use cases (illustrative)

#### UC1 — Verification email content includes correct link

- **Preconditions:** Email sending is enabled in the environment
- **Steps:**
  - Sign up with a new email.
  - Capture the received verification email content.
  - Verify the email contains exactly one verification link and it points to the correct domain.
- **Expected results:**
  - Link is present, correct domain, correct path, contains token parameter.

#### UC2 — Verification email renders correctly on mobile clients

- **Steps:**
  - Trigger verification email.
  - Open the email in a mobile email client (or emulator).
  - Ensure the CTA button is visible and clickable; no layout break.
- **Expected results:**

- No broken layout; CTA works; text is readable.

### UC3 — Resend verification does not spam-send multiple emails

- **Steps:**

- Sign up and do not verify.
- Click **Resend verification** repeatedly (e.g., 5 times within 1 minute).
- Check inbox and server response behavior.

- **Expected results:**

- Throttling applied OR only latest email is valid (as per docs).
- Clear user feedback about resend limit.

### Sample Input data:

Sample data source -  Booking.com

Sample single requirement doc -  Dashboard Feature

---

## Notes

- You are free to implement this as a CLI, local web app, or minimal UI.
- Prefer **clean architecture** (ingestion module, retrieval module, generation module, safety/guard module).
- The best submissions are easy to run, explainable, and hard to break.
- You can use any coding IDEs if that will speed up your development.

- If you need access to any of the **LLM models**, please reach out via LinkedIn message -  
<https://www.linkedin.com/in/santhosh-selladurai-bb601b49/>. (only limited access on select models can be provided)