**Coding Exercise (20 min)**

**Title**

Document Ingest: Validate and Forward

**Problem statement**

You’re implementing a component in a **protocol-agnostic document ingest pipeline**. Uploads may arrive via HTTP, gRPC, CLI, etc. For this task, you are given:

* A **finite, read-once byte source** representing an uploaded file. Treat it like a request body: once you read bytes, you **cannot re-read** them.
* **Request metadata** provided by the uploader (which may be wrong or incomplete).
* A **sink** you call to forward the upload and its result for downstream processing.

Implement the ingest component to:

1. **Consume** the provided byte source and **compute**:
   * detectedMime (by inspecting content),
   * size (total bytes consumed),
   * sha256 (hex digest).
2. **Validate** using both request metadata and static configuration:
   * If contentLength is provided, require size == contentLength.
   * Enforce a static maxContentLength.
   * Enforce a static allowlist acceptedMimes (type/subtype; ignore params).
3. **Forward** the upload to the given **sink**, along with your computed result and original metadata.
4. **Return** void from the ingest call (the sink is authoritative for persistence).

Context: you’ll receive three test files (PDF, DOCX, PNG). Some uploads may omit contentLength (e.g., live capture), but the total is finite.

**What you may use**

* Any **statically typed** language (Scala / Java / Kotlin / C# / Rust / C++).
* Any editor/IDE and **any AI assistant**.
* Standard libraries + a minimal test framework.

**Inputs (provided at runtime)**

* filename: String
* claimedMime: String (may be wrong)
* contentLength: Option[Long] (may be absent)
* bytes: a **finite sequence or stream of bytes** (non-rewindable)
* **Config (constant for the run):**
  + maxContentLength: Long
  + acceptedMimes: Set[String]
* **Test files:** sample.pdf, sample.docx, sample.png

**Result model (for validation + reporting)**

type IngestResult = {

detectedMime: String,

size: Long,

sha256: String,

ok: Boolean,

errors: List[String] // empty if ok

}

**Conceptual interfaces (language-agnostic)**

The following are **guides**, not required signatures.  
You may adapt them to whatever abstractions make sense in your chosen language (e.g., InputStream, Iterator<byte[]>, IAsyncEnumerable<byte[]>, or functional Stream[Chunk[Byte]]).

type UploadMeta = {

filename: String,

claimedMime: String,

contentLength: Option[Long]

}

type IngestConfig = {

maxContentLength: Long,

acceptedMimes: Set[String]

}

// Represents a finite, read-once sequence of bytes.

interface ByteSource {

// Pulls the next chunk or returns an empty/EOF marker.

// May throw on I/O error.

nextChunk(): ByteChunk

}

// The sink receives your result and the same bytes you validated.

// Implementations in tests will just consume and count bytes.

interface IngestSink {

persist(meta: UploadMeta, result: IngestResult, data: ByteSource): void

}

// Your ingest component: performs validation and forwards to the sink.

interface Ingestor {

ingest(meta: UploadMeta, cfg: IngestConfig, src: ByteSource, sink: IngestSink): void

}

**Note:** feel free to redesign these types (e.g., use an iterator, a stream API, or async generators).  
What matters is that:

* The byte input is **finite and non-rewindable**.
* You can iterate, read, or stream bytes to compute the hash and validations.
* The same bytes are forwarded to the sink.

**Functional requirements**

* **Content-based MIME detection** (inspect bytes, not just filename).
* **Compute** size and sha256.
* **Validate**:
  + If contentLength is present → require exact match.
  + Reject if size > maxContentLength.
  + Reject if detectedMime ∉ acceptedMimes.
  + Aggregate all validation failures in errors; set ok = errors.isEmpty.
* **Forwarding**: call the sink once, providing a byte source or iterator that represents the **same bytes** you validated.
* **Failure handling**: sensible exceptions on I/O errors; ensure resources are cleaned up.

**Tests (write a few quick ones)**

* **Happy paths**: pdf, docx, png → correct detectedMime, size, sha256, ok = true.
* **Negative**:
  + Wrong claimedMime (but detectedMime accepted) → record mismatch error (still ok if policy allows).
  + contentLength off by ±1 → error recorded.
  + Exceeds maxContentLength → error recorded.
* **Edge**:
  + contentLength = None → still enforce maxContentLength.
  + Tiny/empty input → sensible result and errors if applicable.
* **Mock sink**:
  + Provide a test IngestSink that **consumes all bytes** and **counts them**; returns void.
  + Assert the sink observed exactly result.size bytes and did not exceed maxContentLength.

**Evaluation (what we’re looking for)**

* **Correctness (3 pts)**: MIME by content; accurate size & SHA-256; validations applied.
* **I/O judgment (3 pts)**: sensible handling of a finite, non-rewindable source (no obvious memory foot-guns).
* **API & clarity (2 pts)**: clean types, readable code, clear error messages.
* **Tests (1 pt)**: happy/negative/edge + mock sink verifying byte consumption.
* **Resilience (1 pt)**: error handling and cleanup.