Http inventory

Thursday, 19 February 2026 12:09 PM

localhost:8081/endpoint

Ingestion Service
- middleware check
- data structure validation
- create metadata prep
- ETL processing (if req)
- DB writes

Write Aside Caching: lazy caching

Reason: ① cache independence ② fault tolerance ③ cache downtime & loss of data

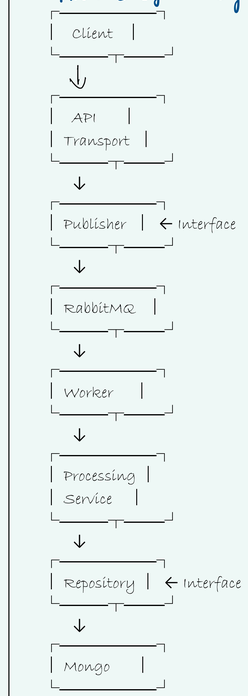
2nd priority will add if time permits

Response is independent of successful DB storage

Response types
1. 200 ok – successfully ingested
2. error response in case of failure in ingestion
   - eg:-> capacity of queue is reached.
   - -> queue unavailability

Assumptions
1. Data Consistency is of prime importance
2. Data Availability is 2nd priority { older data might be fetched from cache. }
3. A cache invalidation API can be setup but not advisable post every DB update
4. Cache TTL can be set to 1 hour { ensuring everything before that is updated correctly. }

Final Design – Ingestion



Design Schema (rough)

```
{
  _id: ObjectId,
  url: string,                    // unique
  status: string,
  headers: object | null,
  cookies: object | null,
  page_source: string | null,
  error_message: string | null,
  retry_count: int,
  created_at: datetime,
  updated_at: datetime,
  last_attempt_at: datetime | null
}
```

cons:
1. model modifications would be needed in future if any new data needs to be scraped
2. not extensible
3. too many fields in model.

4 Scp Containers in Docker.

① API
- health endpoints
- post
- get
- exp backoff retries to connect with queue & DB

② worker
- exp backoff retries to connect with queue & DB
- manage states
  - consume msgs
  - retry logic
  - update in DB

③ DB
MongoDB volume mount

④ RabbitMQ.
- Durable
- persistent
- volume mounted (disk)
- prefetch = 1
- max length = 1000
- Default priority (for MVP no retry 😉)
- reject publish if overflow

API

client ⟷ Publisher

Response types
1. Accepted 202 → safely enqueued
2. 503 → publisher not ready → queue overflow → broker down
3. 400 → empty / invalid URL
4. 500 → unexpected exception (extra caution)

Stable Transitions

Publish → QUEUED ①

QUEUED → IN_PROGRESS ②

IN_PROGRESS → COMPLETED ③

IN_PROGRESS → FAILED_RETRYABLE (retry < MAX) ④

IN_PROGRESS → FAILED_PERMANENT (retry >= MAX) ⑤

Processing Flow

Worker

On message receive:

1. Upsert record (if not exists)
2. Transition → IN_PROGRESS
3. Fetch metadata
4. If success:
   - Update status → COMPLETED
   - Store headers, cookies, page_source
   - ACK
5. If failure:
   - Increment retry_count
   - If retry_count < 3:
     - Update status → FAILED_RETRYABLE
     - NACK (requeue)
   - Else:
     - Update status → FAILED_PERMANENT
     - ACK

DB Schema design

Indexes:
1. URL ⇒ **** for faster query
2. Created_at ⇒ for querying in date range

Final Schema

```
{
  "_id": ObjectId,

  "url": "string",
  "status": "QUEUED | IN_PROGRESS | COMPLETED | FAILED_RETRYABLE | FAILED_PERMANENT",

  "metadata": {
    "headers": {...} | null,
    "cookies": {...} | null,
    "page_source": "string" | null
  },

  "processing": {
    "retry_count": int,
    "error_message": "string" | null,
    "last_attempt_at": datetime | null
  },

  "additional_details": {...} | null,

  "created_at": datetime,
  "updated_at": datetime
} s
```