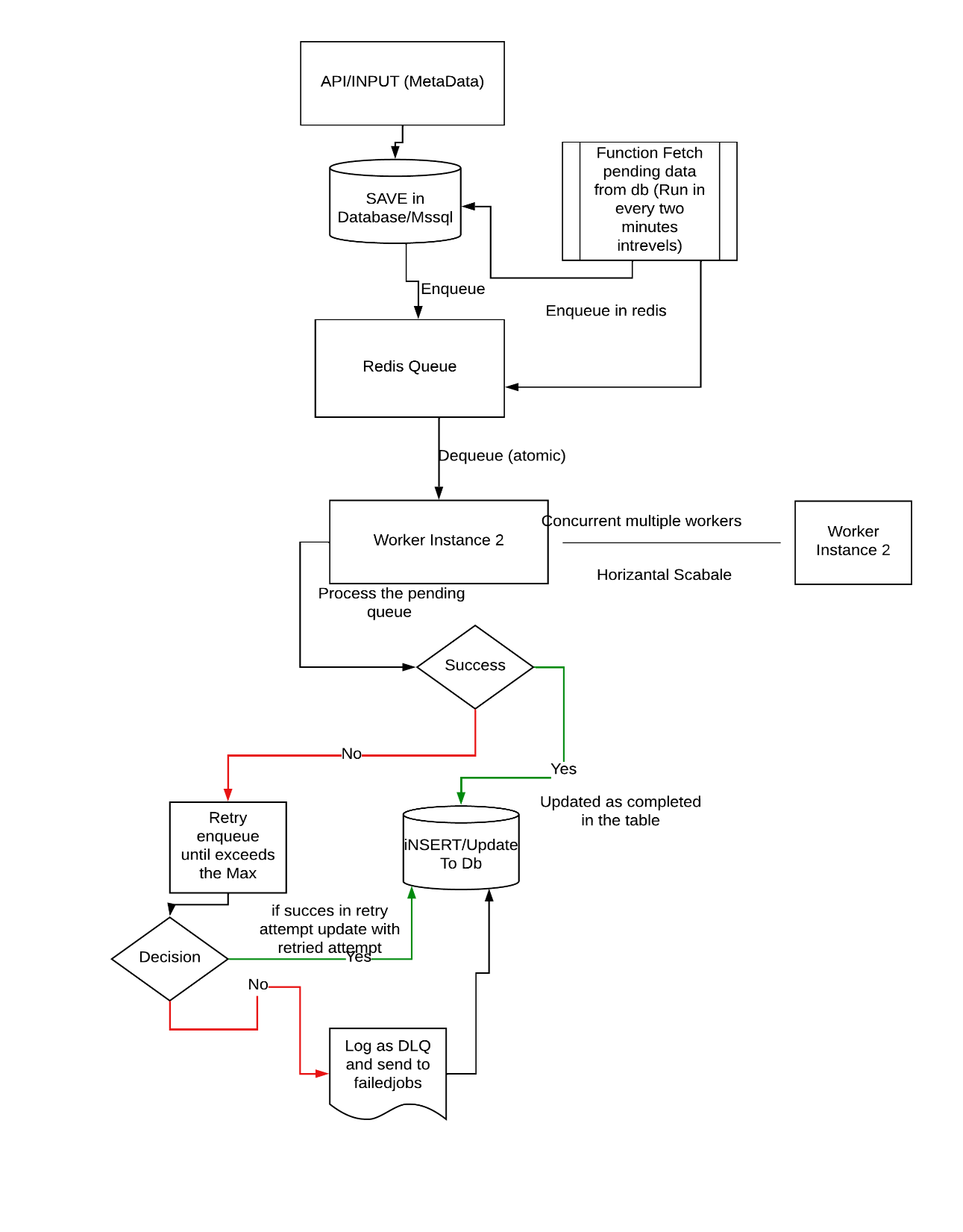
System Work Flow



**High-Level Implementation (with Component Responsibilities)**

**1. Job Submitter (Future Scope / API Layer)**

* Accepts job creation requests (e.g. email, export task)
* Validates input
* Inserts into Jobs table with Status = 'queued'

**2. Scheduler Engine (main.go)**

* Loads all PENDING jobs from DB
* Puts them into appropriate queue based on priority
* Starts multiple workers (goroutines) to process jobs concurrently

**3. Job Queue (queue/queue.go)**

* Maintains 3 internal queues:
  + High (0)
  + Medium (1)
  + Low (2)
* Implements Enqueue() and Dequeue() with priority handling
* Fair fallback logic: If High is empty → check Medium → then Low

**4. Worker Pool (worker/worker.go)**

* Goroutines continuously pull jobs from queue
* Simulate job processing (replace with actual logic)
* If success → update job status to COMPLETED
* If failure:
  + If retries left → requeue job
  + If exhausted → send to DLQ table and mark job as FAILED

**5. Database Layer (jobs/repository.go, jobs/dlq.go)**

* operations for:
  + Loading jobs from DB
  + Updating job status
  + Updating retry count
  + Sending to DLQ (inserts into FailedJobs table)

**Setup & Running Instructions**

| **Component** | **Technology / Library** | **Purpose** |
| --- | --- | --- |
| Programming Language | Go (Golang) | Main backend language |
| Redis Client | github.com/redis/go-redis/v9 | Redis interaction for queue ops |
| Database Driver | database/sql + pq / mssql | Persistent job data storage |
| JSON Serialization | encoding/json | Job data marshaling |
| UUID Generation | github.com/google/uuid | Unique job IDs |
| Logging | Go std log package | Logging system events |

Required Packages

| **Package** | **Install Command** | **Purpose** |
| --- | --- | --- |
| **MSSQL Driver** | go get github.com/denisenkom/go-mssqldb | Connect to Microsoft SQL Server from Go |
| **UUID Generator** | go get github.com/google/uuid | Generate unique IDs for jobs |
| **Redis Client** *(if using Redis)* | go get github.com/go-redis/redis/v9 | Interact with Redis for job queues |
| **Env File** | go get github.com/joho/godotenv | To get value from .env file |

If you have docker installed on your machine then you test directly by using command

**docker-compose up --build**

### Prerequisites

* Go (version 1.18+ recommended)
* Redis server (running locally or accessible)
* MSSQL server with the provided tables
* Set environment variables or modify connection on **.env** file
* **Docker** For scaling (Optional)

Configure Database

Paste the .Sql File and execute in the MSSQL. it is present in the SQL\_File folder you can check in the repo and there are also some sample data to insert in the table.

### Start Redis Server

Make sure Redis is running and accessible at localhost:6379 or update internal/queue.go accordingly to start redis port runner.

To Run the Project run this command

**go run cmd/main.go (in the terminal)**

Api URL to Post the metadata

Method (Post)

http://localhost:8080/enqueue

I have also kept the sample **Json Paylod** on the file called **rest-Api-payload-sample.json** in the Repo you can use that payload for enqueuer method.

note: I am using port **8080** keep the port you are using

and to view the failed DLQ data

Method (Get)

http://localhost:8080/failed-jobs

## **Operational Notes**

* **Deduplication:** Jobs have unique IDs and are deduplicated on enqueue.
* **Retries & DLQ:** Jobs that fail are retried up to MaxRetries. Failed jobs go to DLQ and logged in DB.
* **Atomic Dequeue:** Redis RPOP ensures atomic removal of jobs for workers.
* **Priority:** Current implementation supports priority as an integer but uses a single queue — can be extended.