# 🔄 Fallback & Retry Queue System

## ✅ Module Overview

The Fallback & Retry Queue System is a core infrastructure module designed to ensure reliable execution of background tasks and third-party API operations across the SaaS platform. It captures failed jobs (e.g., blog generation, Facebook post failure, product sync error), logs context, and retries them based on priority and backoff strategy. It guarantees delivery, even during temporary API failures, rate limits, or platform downtimes.

## 🎯 Goals

* Guarantee task execution and message delivery
* Decouple background jobs from user flow
* Automatically retry failed jobs with smart rules
* Allow superadmins to inspect, pause, or replay queued jobs

## 🧠 Core Features

### 1. **Persistent Job Queue Engine**

* **Backed by Redis + RDBMS fallback**
* Queues types:
  + post\_publishing
  + blog\_generation
  + customer\_sync
  + ugc\_tracking
  + email\_sms\_campaign
  + image\_generation
* Supports FIFO and priority-based queues

### 2. **Smart Retry Logic**

* Retry Policies:
  + Exponential backoff (e.g., 10s, 30s, 1m, 5m…)
  + Max retries: configurable per job type
  + Alert thresholds (e.g., alert after 3 fails)
* Fallback paths:
  + Switch API keys (in case of quota limits)
  + Use backup generation service (e.g., switch GPT provider)

### 3. **Job Status Dashboard**

* Real-time admin view:
  + Pending, in-progress, succeeded, failed
  + Execution time, retry count, next retry ETA
* Filters:
  + By job type, brand, module, status
* Action buttons:
  + Re-run, Pause, Cancel, Force Execute

### 4. **Error Context Logging**

* Logs the exact payload, API response, stack trace
* Saves failed job data for inspection and debugging
* Linked to audit trail and brand-level usage history

## 🔧 Technical Architecture

### Database Tables

* job\_queue
  + job\_id, type, brand\_id, payload, status, attempts, next\_retry\_at, created\_at
* job\_logs
  + log\_id, job\_id, error, api\_response, timestamp, trace

### Services

* **Job Scheduler Worker** (runs every 5s)
* **Retry Handler** (triggers re-attempts)
* **Error Notifier** (sends alert to superadmin on repeated failure)
* **Queue Monitor API** (UI integration)

## 🔐 Permissions & Roles

* **Superadmin**:
  + View/edit all queues
  + Force execute jobs
  + Change retry logic globally
* **Brand Admins**:
  + View only brand-level job queue
  + See logs and retry failed jobs (within limit)
* **DevOps**:
  + Debug failures
  + Patch failing modules from log context

## 📊 Metrics to Track

* Total tasks per module per day
* Success vs failed ratio
* Avg retries per task
* Longest delay queue item
* Jobs requiring fallback

## 🔄 System Integrations

* Facebook/Instagram API syncs
* AI engines (OpenAI, image generation)
* Blog auto-posting engine
* WooCommerce/Shopify sync
* Email/SMS campaign sender
* UGC processing jobs

## 🧪 Example Jobs

**Post Generation Retry Job**

{  
 job\_id: 12881,  
 type: "post\_publishing",  
 brand\_id: 103,  
 payload: {  
 platform: "facebook",  
 caption: "Sale Alert!",  
 image\_url: "https://...",  
 scheduled\_time: "2025-07-20T10:00"  
 },  
 status: "failed",  
 attempts: 2,  
 next\_retry\_at: "2025-07-20T10:30"  
}

**Retry Logic:** 3 attempts → alert + fallback to backup API

## 🧠 Benefits

* Makes platform resilient to 3rd party failures
* Gives control to debug + retry important workflows
* Supports eventual consistency
* Reduces need for real-time failures to reach the end-user

✅ Module ready. Let me know if you want to visualize the Retry Queue UI next or move to the next module.