📄 Product Requirements Document (PRD)

⚡ AsyncFlow Hub (NestJS Edition)

🧭 1. Overview

Product Name

AsyncFlow Hub

Summary

AsyncFlow Hub is a backend system that offloads long-running operations into background jobs using a queue-based architecture built with NestJS, Redis (BullMQ), and PostgreSQL.

Instead of executing heavy tasks inside HTTP requests, the system processes them asynchronously using workers.

🎯 2. Problem Statement

Modern backend systems face challenges such as:

- Slow API responses due to heavy operations
- Tight coupling between business logic and request lifecycle
- Poor scalability under high traffic
- Lack of visibility into long-running tasks
- Repeated implementation of background processing logic

Core Problem

> Backend systems need a reliable way to execute long-running tasks without blocking user requests.

💡 3. Proposed Solution

Build a unified asynchronous job processing platform where:

- API layer only creates jobs
- Redis queue manages job distribution
- Workers process jobs independently
- System tracks job lifecycle end-to-end

🏗️ 4. System Architecture

```
Client
  ↓
NestJS API
  ↓
BullMQ Queue (Redis)
  ↓
NestJS Workers (Processors)
  ↓
PostgreSQL / Email Service / Storage
```

🔑 5. Core Features

5.1 Job Queue System

Central system for managing background tasks.

Requirements:

- Enqueue jobs via API
- Support multiple job types
- Retry failed jobs automatically
- Dead-letter queue for failed tasks

5.2 File Processing Pipeline

Handles uploaded files asynchronously.

Workflow:

1. User uploads file
2. Job is created: `process_file`
3. Worker processes file:
4. resize images
5. compress files
6. generate thumbnails
7. Store processed output

5.3 Report Generation Engine

Generates heavy reports in background.

Workflow:

1. User requests report
2. Job created: `generate_report`

3. Worker:

4. fetch data from DB

5. aggregate results
6. generate PDF/CSV
7. Store and return report link

5.4 Notification System

Event-driven communication layer.

Events:

- user registration
- file processed
- report completed

Job Types:

- send_email
- send_notification

5.5 Scheduled Jobs

Supports recurring background tasks.

Examples:

- daily cleanup jobs
- weekly report generation
- retry monitoring jobs

5.6 Job Tracking System

Tracks full lifecycle of each job.

States:

- queued
- processing
- completed
- failed
- retrying

🔄 6. Job Lifecycle

```
Created → Queued → Processing → Completed
                       ↓
                   Failed → Retry → Dead Letter Queue
```

🧱 7. Data Models

User

- id
- email
- password
- created_at

Job

- id
- type
- payload
- status
- retry_count
- max_retries
- created_at
- updated_at
- result
- error_message

File

- id
- user_id
- original_url
- processed_url

- status

Report

- id
- user_id
- type
- file_url
- status

📡 8. API Requirements

Auth

- POST /register
- POST /login

Jobs

- POST /jobs
- GET /jobs/:id
- GET /jobs?status=

Files

- POST /upload
- GET /files/:id

Reports

- POST /reports
- GET /reports/:id

⚙️ 9. Non-Functional Requirements

- API response time < 200ms (excluding async tasks)
- At-least-once job execution
- Horizontal worker scalability
- Reliable retry mechanism
- Centralized logging system

- Observability for job states

🧠 10. Design Principles

- **Idempotency** → safe retries without duplication
- **Decoupling** → API separated from workers
- **Fault tolerance** → retries + dead-letter queue
- **Small tasks** → each job does one responsibility
- **Event-driven design** → everything is triggered via events

---

📊 11. Success Metrics

- ≥ 95% job success rate
- < 5s average queue delay
- 70% reduction in API latency for heavy tasks
- Zero data loss during retries
- Scalable worker performance under load

🚀 12. MVP Scope

Phase 1

- Job queue system
- Basic worker implementation
- Job tracking system

Phase 2

- File processing pipeline
- Notification system

Phase 3

- Report generation engine
- Scheduled jobs

🌱 13. Future Enhancements

- Workflow DAG system (job chaining)
- Priority queues
- Real-time job dashboard (WebSockets)
- Multi-tenant architecture

- Distributed worker clusters
- Admin observability dashboard

🧠 14. Key Insight

AsyncFlow Hub transforms backend design from:

> Request-driven execution

to

> Event-driven distributed job processing