Coding Challenge
The challenge completion time is one week.
Task: Simplified Distributed File Processing API
Objective:
Create a simplified version of a distributed file processing system. Users will be able to upload files, which are processed asynchronously by worker nodes. The file processing task is simplified to simulate a CPU-intensive task (e.g., parsing file contents or basic data transformations). The system should handle multiple concurrent uploads, task queuing, and progress tracking.

Project Requirements:
1. User Authentication:
- Implement user registration and login with JWT authentication.
- Ensure that only authenticated users can upload files.

2. File Upload and Storage:
- Allow users to upload files up to 100MB.
- Store files locally (for simplicity) or use a cloud solution like AWS S3 if you prefer.
- Implement chunked file uploads for network resilience (optional, for added complexity).

3. File Processing Simulation:
- Once a file is uploaded, simulate a CPU-intensive task (e.g., reading file contents, performing a fake "data transformation" like capitalizing all text).
- Use worker threads in Node.js to process files asynchronously.
- Each worker should process only one file at a time.

4. Task Status Tracking:
- Provide an endpoint for users to check the status of their file processing tasks (e.g., "in progress," "completed," or "failed").
- Use a basic in-memory store (like Redis or in-memory database) to track the status of each task.

5. Notifications (Basic):
- Implement basic progress tracking using polling (users can periodically hit the API to check task status).
- Optionally, add WebSocket or Server-Sent Events (SSE) to notify users in real-time when their task is completed.

Additional Complexity (Optional Enhancements for Ambitious Candidates):
1. File Processing Timeout and Retry:
If a task takes longer than a specified time limit (e.g., 2 minutes), mark it as failed and retry up to 3 times before giving up.

2. Concurrent Task Processing:
Ensure that multiple tasks can be processed concurrently, but limit the maximum number of concurrent tasks to avoid overwhelming the system (e.g., max 2 tasks at a time).

Technical Requirements:

- **Backend Framework:** Use Node.js with Express.
- **Task Queue:** Use a simple queue system (in-memory queue or a library like Bull or Kue).
- **Database (Optional for Task Tracking):** Use Redis or an in-memory data store to track task statuses.
- **Testing:** Write unit tests for key features (file upload, task processing, task tracking).
- **Documentation:** Provide basic API documentation (e.g., via Swagger or Postman).

Evaluation Criteria:

- **Scalability and Concurrency Handling:**
  Does the system handle multiple concurrent uploads and tasks without performance issues?
- **Code Quality:**
  Is the code clean, modular, and easy to understand? Does it follow best practices?
- **Task Management and Fault Tolerance:**
  Are tasks managed efficiently, and does the system handle failures gracefully (e.g., timeouts, retries)?
- **Testing:**
  Are critical components covered by unit tests, especially file uploads and task processing?

Submission Guidelines:

- **Time Estimate:** Around 5 hours to complete.
- Submit the code via a GitHub repository link.
- Include a README file explaining how to run the system locally.
- Ensure the project is functional, testable, and easy to run (consider using Docker for quick setup if applicable).