# **Build a Data Ingestion API System**

We’d like you to design and implement a simple API system to test your skills in building APIs and incorporating basic logic. Your task is to create two RESTful APIs: one for submitting a data ingestion request and another for checking its status. The system should fetch data from a simulated external API, process it in batches asynchronously, and respect a rate limit.

### **Requirements**

1. **Ingestion API**
   * **Endpoint**: POST /ingest
   * **Input**: A JSON payload containing a list of IDs (integers) and Priority.  
     + ids → list of integers → id can be in the range of (1 to 10^9+7)
     + priority → Enum → (HIGH, MEDIUM, LOW)
     + Example: {"ids": [1, 2, 3, 4, 5], "priority": 'HIGH'}
   * **Behavior**:  
     + You can process ONLY 3 id’s at any point in time.
     + Enqueue each batch as a job to be processed asynchronously.
     + Simulate fetching data for each ID from an external API (no real API calls needed—just mock the behavior with a simple delay and return a static response like {"id": <id>, "data": "processed"}).
     + Respect a rate limit of **1 batch per 5 second**. (ie: max 3 ids per 5 second)
     + If you get another request with higher priority, those id’s should be processed before the lower priority ids
     + you should process the data based on (priority, created\_time)
     + Example
       - Request 1 - T0 - {"ids": [1, 2, 3, 4, 5], "priority": 'MEDIUM'}
       - Request 2 - T4 - {"ids": [6, 7, 8, 9], "priority": 'HIGH'}
         * T0 to T5 it should process 1, 2, 3
         * T5 to T10 it should process 6, 7, 8
         * T10 to T15 it should precess 9, 4, 5
   * **Output**: Return a unique ingestion\_id immediately as a JSON response.  
     + Example: {"ingestion\_id": "abc123"}
2. **Status API**
   * **Endpoint**: GET /status/<ingestion\_id>
   * **Input**: The ingestion\_id returned by the ingestion API.
   * **Behavior**: Retrieve the processing status of the ingestion request.
   * **Output**: A JSON response showing the overall status and details of each batch.

Example:  
  
 {

"ingestion\_id": "abc123",

"status": "triggered",

"batches": [

{"batch\_id": <uuid>, "ids": [1, 2, 3], "status": "completed"},

{"batch\_id": <uuid>, "ids": [4, 5], "status": "triggered"}

]

}

* + - Possible each level batch status:
      * yet\_to\_start, triggered, completed
    - Possible Outer Status:  
      * If all the status is yet\_to\_start, it will be yet\_to\_start
      * If atleast one is triggered, it will be triggered
      * If all the status is completed, it will be completed.

1. **Technical Constraints**
   * Use any programming language or framework you’re comfortable with (e.g., Python with Flask/FastAPI, Node.js with Express, etc.).
   * Persist the ingestion and batch statuses (e.g., in-memory store, database, or file system) so the status API can retrieve them.
   * Process batches asynchronously (e.g., using background tasks, queues, or threads).
   * Simulate the external API’s rate limit by ensuring batches are processed no faster than 1 per 5 second.

### **Deliverables**

* A working application hosted on a platform of your choice (e.g., Heroku, Railway, Github Pages etc).
* Source code in a version control repository (e.g., GitHub) with clear instructions for running it locally.
* Basic documentation ([README.md](http://readme.md)) explaining your design choices.
* Test file which tests your endpoints. This test file should be extensive and it should verify If your endpoint is honouring all the rate limits and priorities correctly.

### **Sample Requests and Responses**

**Ingestion API Request**:  
  
 curl -X POST <http://localhost:5000/ingest> -H "Content-Type: application/json" -d '{"ids": [1, 2, 3, 4, 5]}'

**Response**:  
  
 {"ingestion\_id": "abc123"}

**Status API Request**:  
  
 curl <http://localhost:5000/status/abc123>

**Response**:  
  
 {

"ingestion\_id": "abc123",

"status": "triggered",

"batches": [

{"batch\_id": 1, "ids": [1, 2, 3], "status": "triggered"},

{"batch\_id": 2, "ids": [4, 5], "status": "triggered"}

]

}

### **What We’re Looking For**

* **API Design**: Clear, RESTful endpoints with appropriate inputs and outputs.
* **Logic Implementation**: Correct batching, asynchronous processing, and rate limiting.
* **Extensive and bullet proof test file**
* **Code Quality**: Clean, readable code with proper error handling.
* **Documentation**: Brief explanation of your approach and setup instructions.
* Add a screenshot of the test run

### **Notes**

* Focus on the core functionality first (ingestion and status tracking). If time permits, feel free to add features like error recovery or retry logic (optional).
* This is a simulated environment—don’t worry about real external APIs; just mock the behavior.
* Feel free to ask questions if anything is unclear!

### **Submission**

* Share a link to your repository (e.g., GitHub) with your code and documentation.
* hosted URL.
* Comments - provide any necessary instructions.

## **Testing**

We will test your API with our internal test file, which checks for correctness, rate limits and response time etc. **IMPORTANT: DO NOT ADD AUTH TO YOUR API.**

Things to share with college candidates before assignment.

1. They will be building and deploying a service with a couple of endpoints.
2. They have to submit the deployed endpoint and github repo.
3. We’ll run an automated test with the hosted url link you submit. Just submit the base URL, we’ll append the endpoint (\ingest and then we’ll call with our payload and test it)
4. Submission link <https://forms.gle/6HWbbfHywwqeN8Hz5>