# Bankit Backend Engineering Challenge — Transfer System

## Objective

Design and implement a simplified backend system for processing money transfers using multiple providers, queues, and status tracking. This test evaluates your ability to work with asynchronous systems, fallback strategies, and clean code structure.

---

## Project Overview

You are to simulate a backend transfer system for Bankit. The system must accept transfer requests, queue them for processing, attempt delivery through multiple providers with varying success rates, and track the status of each transaction. Logs must be recorded, and the user must be able to query the transfer status.

---

## Functional Requirements

### 1. POST /transfer/initiate

- Accepts a JSON payload with the following fields:
  - `user_id` (string)
  - `amount` (number)
  - `currency` (string)
  - `destination_account` (string)
- Adds the transfer job to a queue with:
  - Attempt count
  - Provider index (starting from 0)

    - Status: `pending`

### 2. GET /transfer/status/:user_id

- Returns the current status of a transfer by `user_id`.
- Status can be: `pending`, `success`, or `failed`

---

## Queue Requirements

- Retry each transfer up to **3 times**.
- If a provider fails, switch to the next provider in the list.
- If all providers fail after all attempts, mark the transfer as `failed`.

---

## Provider Simulation

Implement three mock providers with the following characteristics:

| Provider | Success Rate |
|---|---|
| ProviderA | 70% |
| ProviderB | 90% |
| ProviderC | 95% |

Each provider should randomly succeed or fail based on its configured success rate.

---

## Logging

- Log each transfer attempt to a file `logs/transfer_logs.json`.
- Log format:

```
{
  "user_id": "123",
  "provider": "ProviderA",
  "success": false,
  "timestamp": "2025-06-11T12:00:00Z"
}
```

- Append each log entry to the file.

---

## Testing Requirements

Create automated tests for the following scenarios:

- Successful POST `/transfer/initiate`
- Fallback to another provider when the first fails
- Status returned correctly from GET `/transfer/status/:user_id`

You may use **Jest** or **Mocha/Chai**.

---

## Bonus Points

- Use Bull or another Redis-backed queue instead of in-memory.
- Send webhook or email notification when the transfer is completed.
- Persist job and log data in a database.
- Add a configurable max attempt count via environment variables.

---

## Setup Instructions

1. Clone the repo or create a new Node.js project.

2. Use `express`, `body-parser`, and your preferred testing library.
3. Create mock providers with the described success logic.
4. Run the app with `node app.js`.
5. Run tests with `npm test`.

---

## Example Request

### Initiate Transfer

```
POST /transfer/initiate
Content-Type: application/json

{
  "user_id": "user_001",
  "amount": 1000,
  "currency": "NGN",
  "destination_account": "0123456789"
}
```

### Get Transfer Status

```
GET /transfer/status/user_001

Response:
{
  "user_id": "user_001",
  "status": "success"
}
```

---

## Submission

- Zip and submit the full project directory.
- Ensure it includes:
    - All source code

    - A README with setup instructions
    - Your test cases
    - The log file (can be mock-filled)

---

## Evaluation Criteria

- Code readability and structure
- Error handling and retry logic
- Implementation of queue and fallback
- Accurate status tracking
- Correct use of mock providers
- Testing coverage
- Bonus features (if implemented)

Good luck 🚀