TRACTION
AUTOMATING PERFORMANCE

Developer - Phase 2

You have 48 hours to complete the test from the moment you receive the email.

In this test you will have to:

- Design and develop distributed applications using the MERN stack (MongoDB, Express.js, React, Node.js).
- Optimize performance and scalability in high-load environments.
- Manage asynchronous communications and event-driven architectures.
- Implement resilience mechanisms in the management of unreliable external services.
- Clearly document architectural and technical decisions.

The test is in English.

Please check that this field contains the same email address used in Step 1. If not, enter it. *

saivinay023@gmail.com

📦 Project Overview

Scenario

You are tasked with creating a **notification management platform** for email and/or SMS notifications, consisting of the following components:

1. **Notification Collector**
   - An API service built with **Express.js** that receives notification requests and queues them for processing.
2. **Processor Service**
   - A **Node.js** worker that reads notifications from the queue, processes them, and sends them to an external simulated service.
3. **Dashboard**
   - A web application built with **Vue, Svelte, Angular, or React** to monitor the status of notifications (sent, failed, queued), with filtering and search functionality.

Provided Repository

You will start from a provided repository that includes the following **Docker architecture** . The software components must be implemented, except for the external simulated service, which is already provided:

- **MongoDB and Redis** for data persistence and queue management.
- **Notification Collector** (/api): Receives requests and places them in a Redis queue.
- **Processor Service** (/processor): Processes notifications from the queue and sends them to the external service.
- **Dashboard** (/dashboard): A React interface for monitoring notification status.
- **Mock API** (/mock-api): A simulated external service with predefined errors and timeouts.

Before starting development, review the repository structure to understand the separation of services: api for handling incoming requests, processor for managing notification processing, and dashboard for monitoring. The mock-api simulates unreliable external services. You are expected to work within this architecture without modifying the mock-api logic.

Simulated External Service

The project includes an already implemented simulated external service (mock-api) with the following behavior:

- **30% chance** of returning a **429 (rate limit)** error.
- **5%** chance of returning to **500 (server error)** .
- **20%** of requests will result in a **5-second timeout** .

⚠️ **You must integrate this service in a resilient manner without modifying its internal logic.**

🛠Technical Requirements

- **Docker, Node.js, MongoDB, Redis.**
- **REST API** with **Express.js** and input validation (eg, using **Joi** or **Zod**).
- Asynchronous communication using **Redis queues** .
- Error handling for unreliable external services (**circuit breaker, retry logic, exponential backoff**).
- **Dashboard** with **WebSocket** or **Server-Sent Events (SSE)** for real-time updates.
- Optimize the system to handle at least **240 requests per minute** .

📝 Instructions

Step 1: Environment Setup

1. Fork the provided repository : `git@github.com:tractiongroup/flawed-messaging-node.git.`
2. Clone the repository locally and set up the Docker environment.
3. Ensure all services (**MongoDB, Redis, mock-service**) are running correctly.
4. Test the mock API with the following command:

bash

```
curl -X POST http://localhost:1337/send

-H "Content-Type: application/json"

-d ''
```

Step 2: Main Tasks

Fix Docker & Configuration

- Resolve any configuration issues in Docker (ports, networking, environment variables).

API Implementation

- Implement the **POST** endpoint: `/api/v1/notifications`
  - Accepts a JSON payload with the following fields:
    - `type` : **email** or **SMS**
    - `recipient` : recipient address/number
    - `message` : notification content
    - `campaign_id` : UUID
  - Validate the incoming data.

  - Queue the request in **Redis** .

Processor Service

- Read notifications from the queue and send them to the **mock external service** .
- Implement **resilient error handling** :
  - **Retry logic** with **exponential backoff** .
  - **Circuit breaker** for managing persistent failures.

Dashboard

- A simple **one-page** interface (no login required).
- Implement **WebSocket or SSE** for **real-time updates** on notification statuses.
- Add filters for **notification type** and **status** (sent, failed, queued).
- **Design is not a priority** ; focus on functionality.

Step 3: Performance Optimization

- Perform load testing (using tools like **Artillery or k6**) to ensure the system can handle at least **240 requests per minute** .
- Monitor resource usage (**CPU, memory**) and optimize response times.
- Implement a **logging system** (eg, **Winston** or **Pino**).

Step 4: Documentation

- **Update README.md** with:
  - Detailed instructions for setting up the environment.
  - Description of the implemented features.
  - Explanation of the technical choices made for resilience and performance optimization.
- **Create a SOLUTION.md** file with:
  - A list of bugs fixed and the debugging methods used.
  - Explanation of optimization strategies adopted.
  - Reflections on potential future improvements.

🔍 Evaluation Criteria

Category	What We're Looking For
Docker	Correct configuration of distributed environments, container networking, environment variables.
Code Quality	Clear, modular code, proper error handling, appropriate use of middleware and libraries.
Performance	System optimization for high loads, efficient use of resources (DB, cache, queues).
Resilience	Implementation of retry logic, advanced error handling (circuit breakers, timeouts).
Scalability	System's ability to handle high request volumes without performance degradation.
Problem Solving	Methodical debugging approach (log analysis, profiling tools, testing).
IU	Responsive, functional dashboard with real-time updates.
Documentation	Comprehensive and clear explanations of the development process and technical decisions.

❓ Questions?

For technical questions, please contact **Enrico** :
📧 **enrico@tractiongroup.it** / 📞 **+39 328 779 3580**

📤 Submission

Link to your GitHub repository with the project *