Congratulations on making it to the next stage of our interview process! We'd like you to complete the following technical challenge. Please carefully read the updated requirements below, which now include expectations around documentation, configuration, and operational best practices.

🚀 Challenge Overview

A new client has approached us to design and build a backend service for their new platform. The service must pull XML data from public APIs, transform it into JSON, store it in a persistent datastore, and expose it through a single GraphQL endpoint.

Your solution should emphasize **performance**, **maintainability**, **scalability**, and **real-world production engineering principles**.

IMPORTANT:

All code submitted must be authored by **you** without the use of generative AI. You may use documentation sites, manuals, and StackOverflow, but do not use AI-generated code.

🧩 Requirements

1. Data Ingestion

Your service must:

Parse XML from the following endpoints:

- Get all makes:
  https://vpic.nhtsa.dot.gov/api/vehicles/getallmakes?format=XML
- Get vehicle types for a specific make (example):
  https://vpic.nhtsa.dot.gov/api/vehicles/GetVehicleTypesForMakeId/440?format=xml

Transformation Requirements

- Convert all XML to JSON.
- Combine the XML results into a **single unified JSON structure**.
- The final JSON must match the shape shown here:
  https://gist.github.com/mbaigbimm/d340e7800d17737482e71c9ad1856f68

Persist the Result

- Save the fully transformed data to a persistent datastore of your choosing (PostgreSQL, MongoDB, DynamoDB, SQLite, etc.)

2. API Layer

Your service must:

Expose a single GraphQL endpoint

- This endpoint should retrieve the transformed and stored data.
- Queries should be typed, performant, and documented.

3. Testing

Your project must include:

Unit tests

- Cover all data transformation logic.
- Mock external XML API calls.

Integration tests (recommended but optional)

- Cover end-to-end data ingestion → persistence → GraphQL serving.

4. Project Engineering Requirements

TypeScript Required

Docker Required

- Provide a Dockerfile and instructions to run the service.
- Should run locally with minimal setup.

Project Structure & Best Practices

- Follow Node.js and TypeScript industry best practices.
- Use environment-based configuration.
- Maintain separation between domain logic, infrastructure, and presentation.

A. API & Developer Documentation

Your submission must include:

Developer README

- How to set up and run the service locally.
- Environment variable descriptions.
- Build instructions.
- Data model documentation.
- GraphQL schema documentation.
- Example GraphQL queries.

API Documentation

- Describe your internal ingestion pipeline.
- Describe your error handling strategy.
- Document your logging strategy.
- Document your configuration approach.

You may optionally include:

- OpenAPI-style docs for internal service-to-service communication (if applicable).
- Mermaid or sequence diagrams.

B. Error Handling & Logging

Your service must include:

Robust Error Handling

- Graceful failures for network issues.
- XML parsing errors.
- Transformation failures.

- Datastore insertion errors.

Logging

- Structured logging (JSON logs preferred)
- Clearly logged:
    - API request failures
    - Transformation errors
    - Unexpected exceptions
    - Startup and shutdown events

You may use any logging library (Pino, Winston, etc.).

C. Configuration

Your service must:

- Use environment variables for configuration.
- Provide reasonable defaults via a config module.
- Support multiple environments (dev/test/prod).
- Validate configuration on startup (e.g., using Zod, Joi, or similar).
- Document all configuration.

Configuration items should include (but not limited to):

- External API URIs
- Database connection details
- Logging level
- Port number
- Feature flags (optional)

🌟 Nice to Have

- Build the service using **NestJS**.
- Provide a **CI pipeline in github** that:
    - Builds the Docker image
    - Runs linting
    - Runs tests
    - Produces build artifacts or reports

📦 Submission

Please provide a **GitHub repository link** containing:

- Complete source code
- Documentation
- Dockerfile
- Test suites
- Optional CI workflow files

Our team will review your submission within **48 hours** after receiving it.
If you need extra time, just let us know.