Build **Data Ingestion Service** that will handle the ingestion, transformation, and storage of data from multiple sources. It will ensure efficient, secure, and scalable data processing in both batch and real-time modes, and facilitate smooth integration with other systems.

Scope

This Data Integration Service will:

- Ingest data from various sources (files, APIs, databases).
- Transform the data as needed.
- Store the processed data into multiple storage backends (relational databases, NoSQL, cloud storage).

Data Ingestion Service Requirements

1. Functional Requirements

1. Data Source Integration:
    - The service should be able to integrate with multiple data sources (e.g., files, databases, APIs).
    - Support for batch and real-time data ingestion.
    - Provide connectors for common data formats (CSV, JSON, XML, etc.).
2. Data Transformation:
    - Ability to preprocess and transform data before saving or forwarding it.
    - Support for common transformation tasks (e.g., field mapping, data validation).
3. Error Handling:
    - Graceful error handling and logging.
    - Define a retry mechanism for failed ingestion attempts.
4. Data Storage:
    - Ability to ingest and store data into cloud storage or local drive. Data can be ingested in batches or in real-time, and the service should be capable of storing large files (e.g., Parquet, CSV) efficiently in these systems.
    - Optionally, support for cloud storage (e.g., Azure blob Storage).
    - Use of batch processing for large datasets and stream processing for real-time data.
5. Data Monitoring and Auditing:
    - Track data ingestion progress and failures.
    - Provide audit logs for all ingestion operations for traceability.
6. API Support:
    - Expose RESTful APIs to trigger ingestion tasks.
    - Provide endpoints for querying ingestion status and logs.
    - Ability to configure and manage data ingestion jobs through APIs.

2. Non-Functional Requirements

1. Adherence to SOLID principles and best practices to ensure code maintainability.
2. Comprehensive logging and exception handling mechanisms to ease debugging.
3. Unit and integration tests for critical components.
4. The service must be flexible to support additional data sources and formats in the future.
5. Design the system so that adding new transformation or storage strategies can be done with minimal code changes.
6. Programming language Java or Python and choose any suitable framework.