Low Level Design Document

Introduction

This Low Level Design (LLD) document details the implementation plan for **HealthTrends - Distributed Health Data Analyzer**. The project ingests anonymized health records via Kafka, stores them in HDFS, processes them with PySpark, and generates summary CSV reports. The design leverages Python for orchestration and focuses on scalable, distributed big-data processing in the healthcare domain.

1. System Components

Component	Technology	Key Responsibilities
Data Ingestion	Kafka	Receive and queue health records
Data Storage	HDFS	Store raw and processed health data
Data Processing	PySpark	Analyze data, extract trends, generate summaries
Orchestration	Python	Manage workflow, trigger jobs, handle outputs
Output Generation	Python	Write summary reports as CSV files

2. Class/Interface Overview

Class/Script	Responsibility	Key Methods/Attributes
`KafkaProducer`	Sends health records to Kafka	`send(record)`
`KafkaConsumer`	Reads records from Kafka	`consume() -> record`
`HDFSClient`	Reads/writes data to HDFS	`write(data, path)` , `read(path)`
`HealthDataProcessor`	PySpark job for trend analysis	`process(input_path, output_path)`
`ReportGenerator`	Generates CSV summaries	`generate(summary_data, csv_path)`
`Orchestrator`	Controls pipeline execution	`run_pipeline()`

Relationships:

- `Orchestrator` coordinates all components.
- `KafkaConsumer` → `HDFSClient` (raw data).
- `HealthDataProcessor` reads from HDFS, writes processed data to HDFS.
- `ReportGenerator` outputs CSV from processed data.

3. Data Structure Overview

Data Model	Fields / Schema Example
HealthRecord	`patient_id` , `age` , `gender` , `diagnosis` , `timestamp` , …
TrendSummary	`diagnosis` , `count` , `avg_age` , `gender_ratio` , …
CSV Report	Columns: `diagnosis` , `count` , `avg_age` , `gender_ratio`

---

4. Algorithms / Logic

Pipeline Flow (Pseudocode):

```
def run_pipeline():
    # Ingest data
    for record in KafkaConsumer.consume():
        HDFSClient.write(record, raw_data_path)
    # Process data
    HealthDataProcessor.process(raw_data_path, processed_data_path)
    # Generate report
    summary = HDFSClient.read(processed_data_path)
    ReportGenerator.generate(summary, csv_output_path)
```

Trend Extraction (PySpark):

- Group by `diagnosis`
- Aggregate: count, average age, gender ratio

---

5. Error Handling

Scenario	Handling Approach
Kafka connection failure	Retry with exponential backoff, log error
HDFS read/write error	Retry, log, alert if persistent
Data schema mismatch	Validate, skip invalid records, log details
PySpark job failure	Capture exception, log, alert, halt pipeline
Output file write error	Retry, log, alert if unresolved

End of Document