Anomaly Detection for Financial Reconciliation with DeepLearning4J and ServiceNow Integration

1. Introduction

This Low-Level Design (LLD) document provides a detailed technical blueprint for the **Anomaly Detection for Financial Reconciliation** system. The system automates the detection of anomalies in financial data between General Ledger (GL) and iHub systems using a deep learning-based Autoencoder implemented with DeepLearning4J (DL4J). It exposes a REST API for CSV file processing, generates an output CSV with anomaly detection results, and integrates with ServiceNow to log incidents. This document includes class diagrams, sequence diagrams, API specifications, detailed component design, and implementation details to guide developers in building and maintaining the system.

2. System Overview

The system is a Spring Boot application with the following key functionalities:

- **CSV Processing**: Accepts an input CSV file (historical data), processes it using DL4J, and generates an output CSV with anomaly detection results.
- **Anomaly Detection**: Uses an Autoencoder to detect anomalies in real-time financial data based on reconstruction errors.
- **ServiceNow Integration**: Creates an incident in ServiceNow to log the process, retrieving the incident number for traceability.
- **REST API**: Provides an endpoint (POST /api/anomaly/process) for uploading a CSV file and downloading the output CSV.

The system is designed to be scalable, handling potentially millions of records, and is extensible for future enhancements like UI integration and email notifications.

6. Detailed Component Design

6.1 AnomalyDetectionService

- **Purpose**: Handles CSV processing, Autoencoder training, and anomaly detection.
- **Attributes**:
  - autoencoder: MultiLayerNetwork - The DL4J Autoencoder model.
  - featureMeans: double[] - Means of the features for normalization.
  - featureStds: double[] - Standard deviations of the features for normalization.
- **Methods**:
  - processCsv(MultipartFile file): byte[] - Orchestrates the entire process: reads historical data, trains the Autoencoder, detects anomalies, and returns the output CSV as a byte array.

    - readHistoricalData(MultipartFile file): List<String[]> - Reads the input CSV using OpenCSV.
    - trainAutoencoder(List<String[]> historicalRows) - Trains the Autoencoder on historical data.
        - Extracts features (GL Balance, iHub Balance, Balance Difference).
        - Normalizes features using mean and standard deviation.
        - Configures the Autoencoder (3-2-3 layers, ReLU activation, Adam optimizer).
        - Trains for 100 epochs.
    - createRealTimeData(): List<String[]> - Creates hardcoded real-time data (extendable to accept uploads).
    - detectAnomalies(List<String[]> realTimeRows): List<String[]> - Detects anomalies using the Autoencoder.
        - Normalizes real-time data.
        - Computes reconstruction error for each row.
        - Flags anomalies if the error exceeds the threshold (0.1).
        - Adds is_anomaly and anomaly_reason columns.
    - writeOutputData(List<String[]> updatedRows): byte[] - Writes the output CSV to a byte array using OpenCSV.

6.2 ServiceNowService

- **Purpose**: Integrates with the ServiceNow API to create incidents.
- **Attributes**:
    - webClient: WebClient - Spring WebFlux client for making HTTP requests to ServiceNow.
- **Methods**:
    - ServiceNowService(String instanceUrl, String username, String password) - Constructor that initializes the webClient with Basic Auth credentials.
    - createIncident(String shortDescription, String description): Mono<String> - Creates an incident in ServiceNow and returns the incident number.
        - Constructs the request payload with short_description and description.
        - Sends a POST request to the ServiceNow API.
        - Extracts the incident number from the response.

6.3 AnomalyDetectionController

- **Purpose**: Handles HTTP requests and coordinates between services.
- **Attributes**:
    - anomalyDetectionService: AnomalyDetectionService - For CSV processing and anomaly detection.
    - serviceNowService: ServiceNowService - For ServiceNow integration.
- **Methods**:
    - processCsv(MultipartFile file): Mono<ResponseEntity<Resource>> - REST endpoint handler.
        - Validates the input file.
        - Calls anomalyDetectionService.processCsv to process the CSV.
        - Calls countAnomalies to count anomalies in the output.
        - Calls serviceNowService.createIncident to log the process.
        - Returns the output CSV as a downloadable file.

    - countAnomalies(byte[] outputCsvBytes): int - Counts the number of anomalies in the output CSV by checking the is_anomaly column.

---

7. Database Schema

The current system does not use a database, as data is processed in-memory and stored in CSV files. However, for future scalability, a database can be introduced to store historical data, real-time data, and anomaly detection results.

7.1 Proposed Database Schema

- **Table: HistoricalData**
    - id: BIGINT (Primary Key) - Unique identifier for each row.
    - date: VARCHAR(50) - Date of the record.
    - company: VARCHAR(50) - Company identifier.
    - account: VARCHAR(50) - Account number.
    - currency: VARCHAR(50) - Currency type.
    - primary_account: VARCHAR(100) - Primary account description.
    - secondary_account: VARCHAR(100) - Secondary account description.
    - gl_balance: DOUBLE - GL balance.
    - ihub_balance: DOUBLE - iHub balance.
    - balance_difference: DOUBLE - Difference between GL and iHub balances.
    - match_status: VARCHAR(50) - Match status (e.g., "Break").
    - comments: VARCHAR(255) - Comments on the record.
- **Table: RealTimeData**
    - Same structure as HistoricalData, with additional columns:
        - is_anomaly: VARCHAR(10) - Indicates if the row is an anomaly ("Yes"/"No").
        - anomaly_reason: VARCHAR(255) - Reason for the anomaly.
- **Table: Incidents**
    - id: BIGINT (Primary Key) - Unique identifier for each incident.
    - incident_number: VARCHAR(50) - ServiceNow incident number.
    - short_description: VARCHAR(255) - Short description of the incident.
    - description: TEXT - Detailed description of the process.
    - created_at: TIMESTAMP - Timestamp of incident creation.

7.2 Database Integration

- Use Spring Data JPA to define repositories for the above tables.
- Store historical and real-time data in the database instead of CSV files.
- Log incidents in the Incidents table for auditability.

---

8. Implementation Details

8.1 Dependencies

- **Spring Boot**: Version 3.2.4, for building the REST API and dependency injection.
- **DeepLearning4J**: Version 1.0.0-beta7, for implementing the Autoencoder.
- **OpenCSV**: Version 5.7.1, for reading and writing CSV files.
- **Spring WebFlux**: For making asynchronous HTTP requests to the ServiceNow API.
- **Lombok**: For reducing boilerplate code (e.g., getters, setters).

8.2 Configuration

- **application.properties**:

  properties

  CollapseWrapCopy

```
spring.servlet.multipart.max-file-size=10MB

spring.servlet.multipart.max-request-size=10MB

servicenow.instance.url=https://<your-instance>.service-now.com

servicenow.username=<your-username>

servicenow.password=<your-password>
```

- **Constants.java**:
  - Defines Autoencoder parameters (INPUT_SIZE, HIDDEN_SIZE, EPOCHS, LEARNING_RATE, RECONSTRUCTION_ERROR_THRESHOLD).
  - Defines the ServiceNow API endpoint (SERVICENOW_INCIDENT_ENDPOINT).

8.3 Error Handling

- **File Validation**: The controller checks if the uploaded file is empty, returning a 400 Bad Request response if invalid.
- **CSV Processing Errors**: Exceptions during CSV processing (e.g., malformed CSV) are caught and returned as a 500 Internal Server Error with an error message.
- **ServiceNow API Errors**: If the ServiceNow API call fails, the error is logged, and the output CSV is still returned to ensure the core functionality is not disrupted.

8.4 Logging

- Uses SLF4J with Logback (Spring Boot's default logging framework).
- Logs key events:
  - File upload success/failure.
  - Number of anomalies detected.
  - ServiceNow incident creation success/failure.

- Example log:

  text

  CollapseWrapCopy

```
2025-03-26 10:00:00 INFO  ServiceNowService - ServiceNow incident
created: INC0010001
```

---

9. Testing Strategy

9.1 Unit Tests

- **AnomalyDetectionService**:
  - Test readHistoricalData with a sample CSV file.
  - Test trainAutoencoder to ensure the model is trained without errors.
  - Test detectAnomalies with known inputs to verify anomaly detection logic.
- **ServiceNowService**:
  - Mock the WebClient to simulate ServiceNow API responses.
  - Test createIncident to ensure the incident number is extracted correctly.

9.2 Integration Tests

- Test the /api/anomaly/process endpoint with a sample CSV file.
- Verify the output CSV contains the expected columns (is_anomaly, anomaly_reason).
- Mock the ServiceNow API to ensure an incident is created and the incident number is logged.

9.3 Tools

- **JUnit 5**: For writing unit and integration tests.
- **Mockito**: For mocking dependencies (e.g., WebClient).
- **Spring Boot Test**: For testing the REST API and service layer.

---

10. Deployment Considerations

10.1 Environment Setup

- **Java Version**: Java 17 (required by Spring Boot 3.2.4).
- **Build Tool**: Maven (for dependency management and building the application).
- **Server**: Deploy on a server with at least 4GB RAM and 2 CPU cores for small to medium workloads. For large datasets, use a server with GPU support for faster DL4J training.

10.2 Deployment Steps

1. Build the application:

   text

   CollapseWrapCopy

   ```
   mvn clean install
   ```

2. Run the application:

   text

   CollapseWrapCopy

   ```
   java -jar target/anomaly-detection-0.0.1-SNAPSHOT.jar
   ```

3. Access the API at http://localhost:8080/api/anomaly/process.

10.3 Configuration

- Update application.properties with the ServiceNow instance URL, username, and password.
- Adjust spring.servlet.multipart.max-file-size based on expected file sizes.

---

11. Future Enhancements (Detailed)

11.1 Dynamic Real-Time Data

- Add a second file upload parameter to the /api/anomaly/process endpoint for real-time data.
- Update AnomalyDetectionService.createRealTimeData to read from the uploaded file.

11.2 Database Integration

- Use Spring Data JPA to store historical and real-time data in a database (e.g., PostgreSQL).
- Implement repositories for HistoricalData, RealTimeData, and Incidents tables.

11.3 UI Dashboard

- Implement the UI dashboard using React, integrating with the Spring Boot backend via REST APIs.
- Add features like file upload, result visualization, and email notification triggers.

11.4 Performance Optimization

- Implement streaming for large CSV files using OpenCSV’s streaming capabilities.
- Use caching (e.g., Spring Cache) to store trained Autoencoder models for reuse.
- Deploy on a Kubernetes cluster with auto-scaling to handle high traffic.

---

12. Conclusion

This Low-Level Design document provides a detailed technical specification for the Anomaly Detection for Financial Reconciliation system. It includes class and sequence diagrams, API specifications, detailed component designs, and implementation details to guide developers in building the system. The use of Spring Boot, DeepLearning4J, and ServiceNow ensures a robust, scalable, and traceable solution for financial reconciliation. The modular design and comprehensive error handling make the system maintainable and extensible for future enhancements like UI integration, database support, and performance optimization.