Data Agent Governance Tool Design

1. Project Overview

1.1 Background

This project aims to build an automated data governance tool with the following core functionalities:

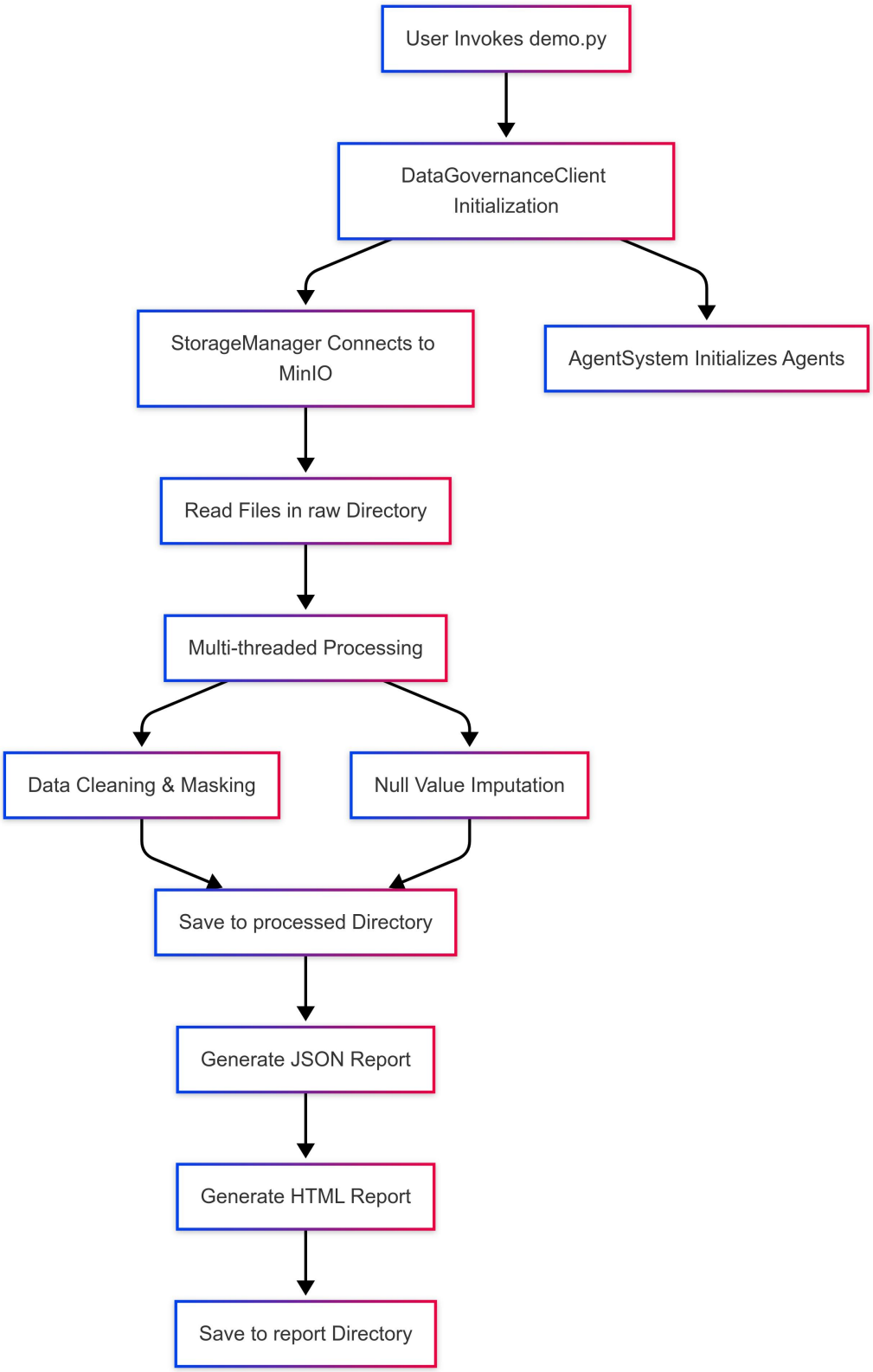
- ✓ Data Cleaning & Masking: Hash sensitive fields and optimize null value imputation.
- ✓ Data Quality Validation: Generate completeness reports and visualizations.
- ✓ Distributed Storage Integration: Seamless integration with MinIO, supporting multiple file formats.
- ✓ Agent Collaboration: Coordinate workflows through multi-role agents (Orchestrator, DataCleaner, etc.).

1.2 Objectives

- ✓ Automate processing of files in the raw directory.
- ✓ Generate structured reports (JSON) and interactive visual reports (HTML).
- ✓ Support high-concurrency processing (via multi-threading).

2. System Architecture

2.1 Architecture Diagram

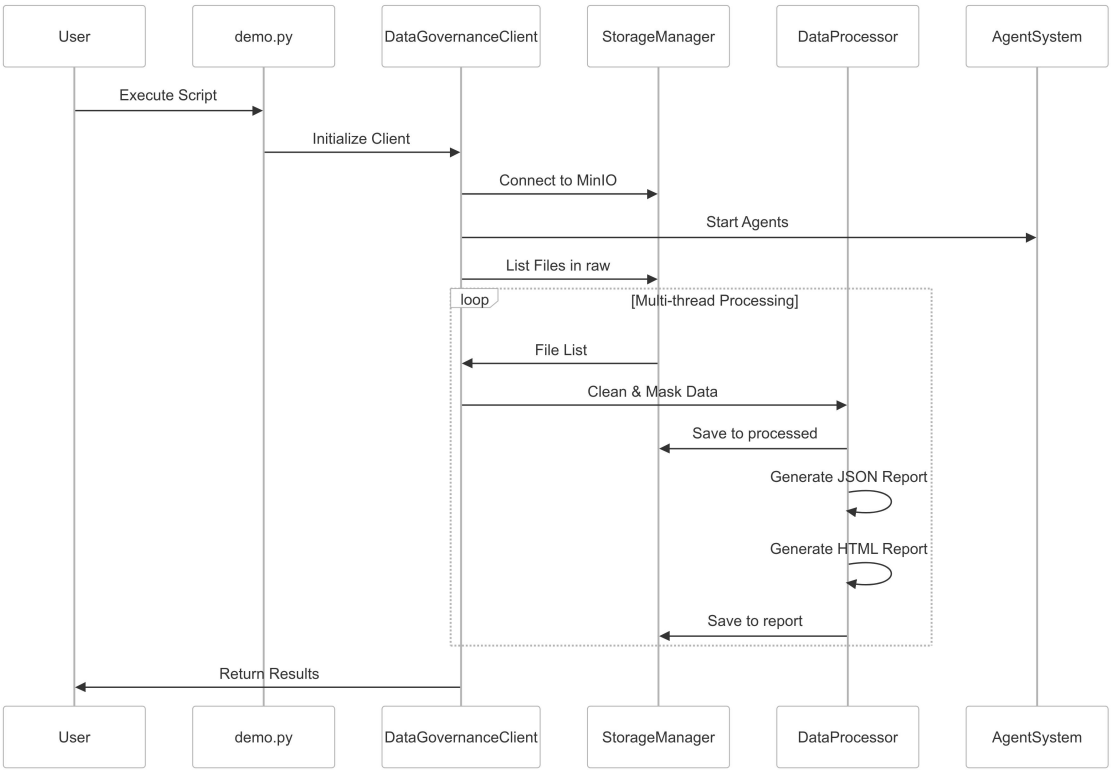


2.2 Core Components

Component	Description
DataGovernanceClient	User interface class coordinating storage, agents, and task distribution.
StorageManager	Encapsulates MinIO operations for file I/O and directory management.
DataProcessor	Handles data cleaning, masking, null imputation, and report generation.
AgentSystem	Manages multi-role agents (Orchestrator, DataCleaner, etc.) via group chat.

3. Execution Workflow

3.1 Flowchart



3.2 Key Steps

3.2.1 Initialization

- ✓ Connect to MinIO via `DataGovernanceClient` and validate permissions.
- ✓ Start the agent system with five roles (Orchestrator, DataCleaner, etc.).

3.2.2 File Processing

- ✓ Use `ThreadPoolExecutor` to process files in the `raw` directory concurrently.
- ✓ For each file:
  - Data Loading: Read CSV/JSON/Parquet files into DataFrame.
  - Field Masking: Truncate and hash sensitive fields (e.g., `name`)
  - Null Imputation: Optimize numeric fields using `IterativeImputer`.

3.2.3 Report Generation

- ✓ JSON Report: Field completeness, top values, and data dimensions.
- ✓ HTML Report: Interactive bar charts using Plotly for completeness analysis.

3.2.4 Result Storage

- ✓ Processed data saved to `processed` directory.
- ✓ Reports saved to `report` directory; error logs written to `logs/errors`.

4. Implementation Details

4.1 Data Masking

```python
# Example: Sensitive field handling
df[col] = df[col].apply(
    lambda x: str(x)[:6] + hashlib.sha256(str(x).encode()).hexdigest()[:6]
)
```

- ✓ Truncate First 6 Chars: Retain partial data for manual verification.
- ✓ SHA-256 Hash: Hash remaining content and take the first 6 digits as a unique identifier.

4.2 Null Imputation

```python
imputer = IterativeImputer()
df[numeric_cols] = imputer.fit_transform(df[numeric_cols])
```

- ✓ Iterative Imputation: Predict missing values using regression models based on numeric field distributions.

4.3 Report Generation

```python
# HTML Report Generation
fig = px.bar(x=fields, y=completeness, title="Data Completeness Report")
html_content = fig.to_html(include_plotlyjs='cdn')
buffer.write(html_content.encode('utf-8'))
```

- ✓ Plotly Interactive Charts: Support zooming and hover-to-view values.
- ✓ CDN Resource Loading: Reduce HTML file size for faster loading.

5. Deployment & Execution

5.1 Dependencies

```bash
pip install pandas minio plotly scikit-learn autogen
```

5.2 Configuration Example

```python
client = DataGovernanceClient(
    minio_endpoint="192.168.1.100:9000",
    minio_access="admin",
    minio_secret="AlexHou@szim",
    ollama_config={
        "model": "deepseek-r1:70b",
        "base_url": "http://192.168.1.101:11434/v1",
        "api_key": "************************"
    },
    bucket="daas-user",
    base_path="alex"
)
```

5.3 Execution Command

```bash
python demo.py
```

6. Scalability & Optimization

6.1 Extensions

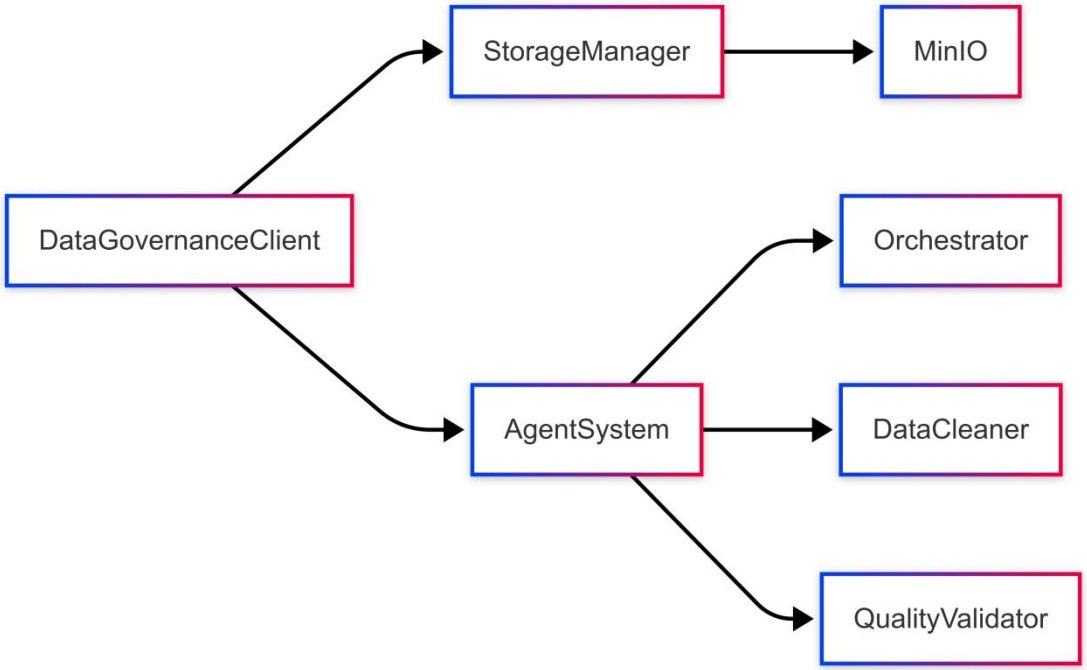
- ✓ Support More Formats: Extend `StorageManager` for additional file types.
- ✓ Custom Masking Rules: Define field-specific masking logic via configuration.
- ✓ Distributed Processing: Integrate Spark/Dask for large-scale data.

6.2 Performance Tuning

- ✓ Caching: Cache preprocessing results in memory or local disks.
- ✓ Async IO: Use `asyncio` to optimize MinIO concurrency.

Appendices

A. Module Relationships



B. Error Handling

- ✓ Exception Trapping: All file operations wrapped in `try-except` blocks.
- ✓ Logging: Errors logged to MinIO's `logs/errors` directory in JSON format.