**Part 1 – Coding**

**Problem Statement:**

Build a system that ingests data from a public API, processes the data, stores it temporarily, exposes it through a RESTful API, and provides a basic front-end interface to display the data.

**Requirements:**

1. **Data Ingestion:**
   * Write a script to fetch data from a public weather API (e.g., USWeather).
2. **Data Processing:**
   * Preprocess and clean the data as necessary (handle missing values, irrelevant fields).
   * Implement a basic data processing step such as basic statistics or aggregation by parameter.
   * Store the processed data in a relational database of your choice.
3. **Back-End API:**
   * Implement a RESTful API using a web framework.
   * Provide endpoints to:
     + Retrieve all data entries.
     + Query data based on certain criteria (date range, value thresholds).
4. **Basic Front-End:**
   * Create a simple web interface using your preferred framework.
   * Display the data fetched from the back-end API.
5. **Local Deployment:**
   * Run the application locally using a local server setup.

**Deliverables:**

1. **Codebase:**
   * A GitHub repository with the complete code for the project.
   * A README.md file explaining the project setup, how to run the code locally, and any dependencies.
2. **API Documentation:**
   * Simple markdown file explaining the API endpoints, request/response formats, and example requests.

**Bonus Points:**

* Cloud deployment - deploy the application on a cloud platform of your choice.
* Enhancing the front-end with additional interactivity or visualizations (behavior over time, data distribution).

**Evaluation Criteria:**

1. **Code Quality:**
   * Clean, readable, and well-documented code.
   * Proper use of version control (e.g., meaningful commit messages).
2. **Technical Skills:**
   * Correct implementation of data ingestion, processing, and storage.
   * Correct integration of front-end with back-end API.
   * Robust and well-documented API endpoints.
   * Functional and user-friendly web interface.
3. **Problem-Solving and Design:**
   * Design considerations of the data pipeline, processing steps, API, and front-end.
   * Effective handling of edge cases and potential issues (e.g., API rate limits, data validation).
4. **Presentation and Documentation:**
   * Clear and concise documentation of the project, including setup and usage instructions.
   * API documentation and front-end usage guide.

**Part 2 – Design**

**Task Overview:** Design a top-level architecture for a system that processes measurement data from a device associated with a labeled sample (referred to as sample X). The system should integrate with external services for quality control and classification, handle user feedback, and store relevant data outcomes.

**Main components:**

1. Data Ingestion:
   * Assume data is stored on a machine to which you have access.
   * Assume the measurement data does not need to be stored permanently but must be available for immediate processing.
2. Quality Control and Classification:
   * The system should interface with an external service that performs quality control on the ingested data.
   * After quality control, the data should be sent to another external service that classifies the sample into one of four classes: A, B, C, or D.
3. User Feedback:
   * Users should be able to confirm the classification or provide an alternative.
   * The system should store the outcome of the classification process, including any user feedback.
4. Data Storage:
   * The QC outcome, classification results and related metadata must be stored securely.
5. Privacy and Access Control:
   * Data is always available only to a user group for which it was generated (parent organisation).
   * Please address use cases where a specific user group’s data needs to be siloed from other user groups vs where no specific limitations beyond user access apply.

**Design Discussion:**

* **Component and interface Choices:** Explain your choices for the components of the system, including data ingestion, processing, and storage, and transitions between them.
* **Technologies:** Recommend specific technologies, frameworks, or services that you would use to implement each component of the system (e.g., message queues, databases, API gateways).

**Output:**

* Provide a diagram that visually represents the top-level architecture of your proposed system.
* Accompany your diagram with a brief written explanation that covers the points mentioned above.

**Additional Considerations:**

* **Resilience:** How does your architecture handle failures, particularly in external services?
* **Microservices vs Monolyth:** Regardless your choice, please discuss briefly on considerations for and against the approaches and why you chose one over the other.