CO2 Monitoring System

Design & Implementation Report/Documentation

Date

Name/ID

1. Introduction

The CO2 Monitoring System is a distributed Java application designed to allow researchers to submit environmental data remotely. The system employs a multi-threaded Client-Server architecture using TCP sockets, ensuring robust data collection and persistence. This report details the architectural decisions, design patterns, and implementation strategies used to meet the project requirements.

2. System Architecture

The system is divided into two main components:

1. **Server:** Listens for connections, manages concurrent clients, and persists data to CSV a file.
2. **The Client:** Provides a user interface for data entry, validates input locally, and transmits data to the server.

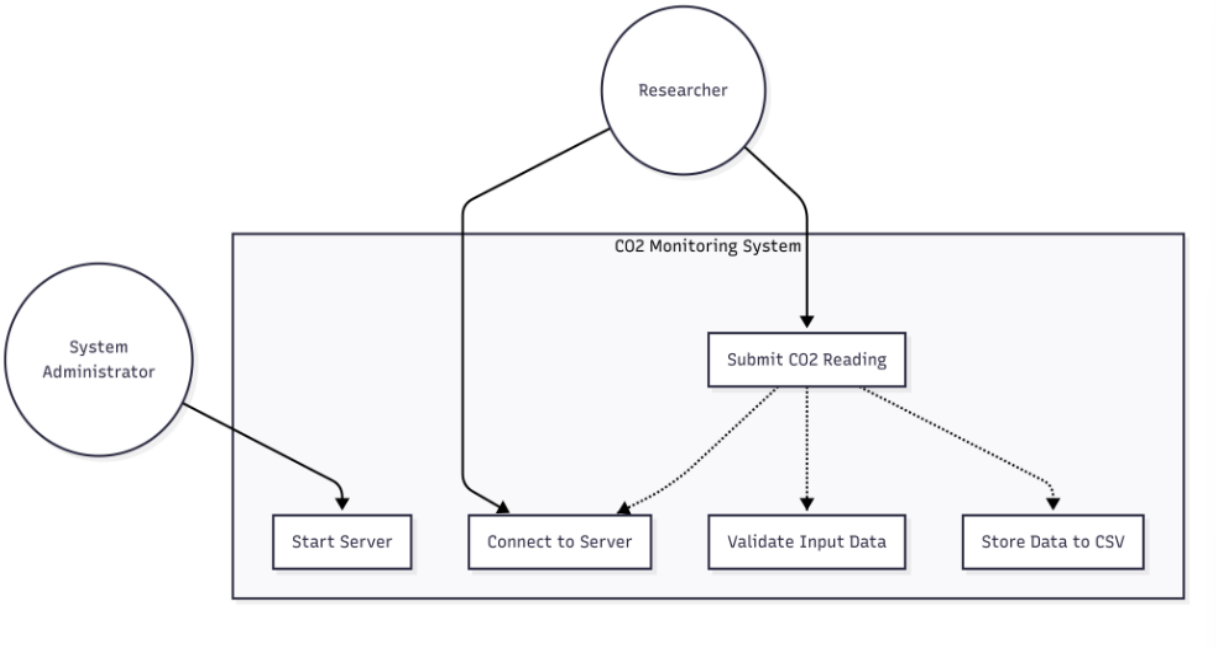

Figure 1. UML Use Case Diagram

3. Design Rationale & OOP Concepts

To ensure scalability, maintainability, and data integrity, specific Object-Oriented principles and design patterns were implemented:

3.1 Concurrency Strategy (Threading)

Instead of creating an unlimited number of threads, which poses a security and resource risk, the system utilizes a Fixed Thread Pool (ExecutorService) limited to 4 concurrent threads. This strictly enforces the requirement to handle "up to four simultaneous clients" without crashing the server under load. The ServerApp acts as the dispatcher, submitting ClientHandler tasks (which implement Runnable) to the pool.

3.2 Data Persistence (Singleton Pattern)

The CSVManager class is implemented using the Singleton Design Pattern. Since multiple client threads need to write to the same records.csv file, having multiple manager instances could lead to file locks or race conditions. A Singleton ensures a single point of access. The critical writeRecord() method is declared synchronized. This ensures that even if four clients submit data at the exact same millisecond, the writes happen sequentially, preventing data corruption.

3.3 Data Integrity (Encapsulation & Immutability)

The Record class serves as a Data Transfer Object (DTO). To prevent accidental modification of research data after submission. All fields (userId, postcode, co2Reading, timestamp) are declared private final [6]. There are no setter methods, making the object immutable once created.

3.4 Input Validation (Abstraction)

Complex validation logic is abstracted into a static Validator utility class. This adheres to the

DRY principle. Both the Client (for user feedback) and the Server (for security) use the exact same logic to validate User IDs and UK Postcodes.

4. Implementation Details

The solution is organized into four packages: server (ServerApp, ClientHandler, CSVManager), client (ClientApp), common (Record), and utils (Validator).

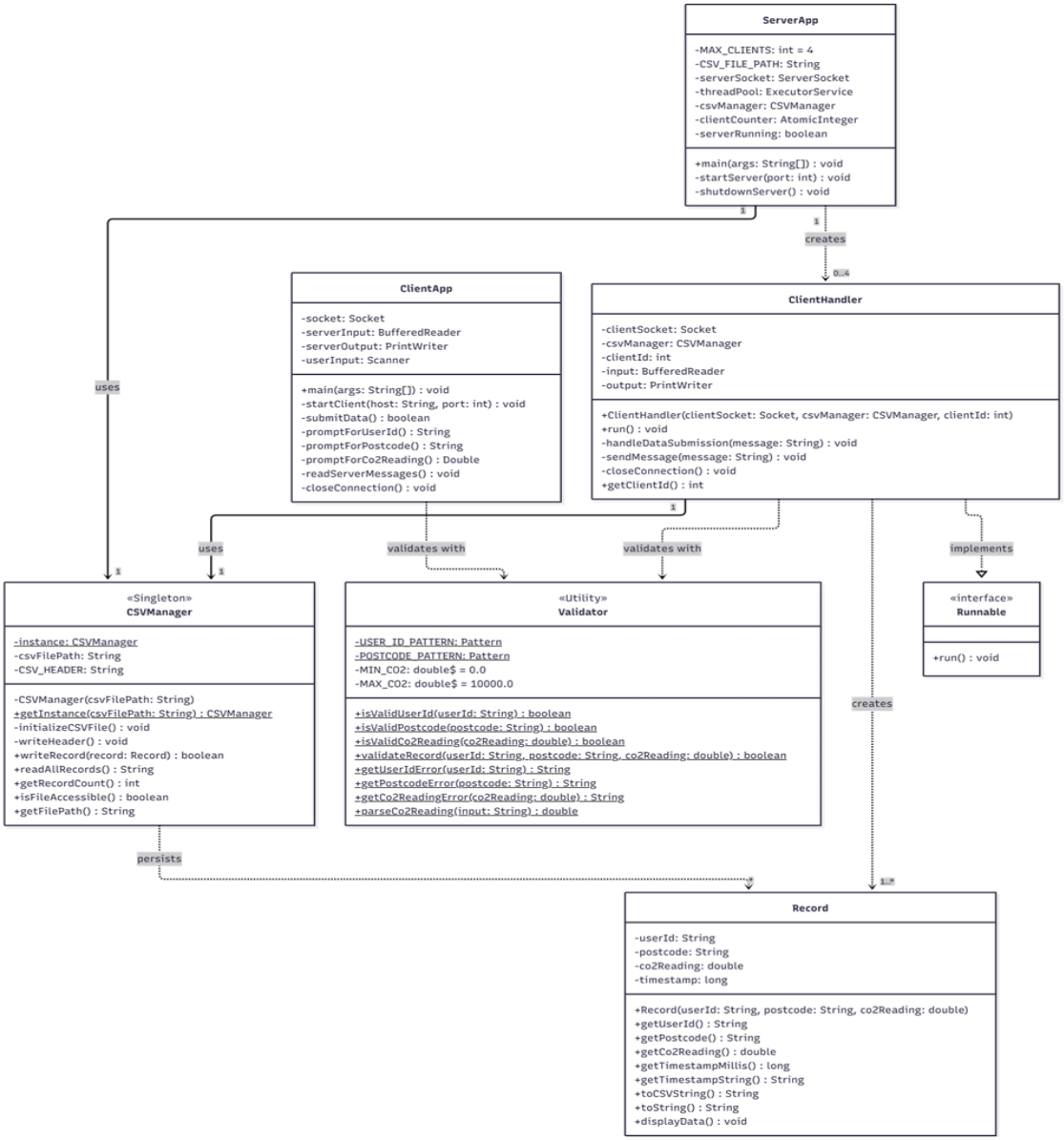

Figure 2. UML Class Diagram

5. Compilation & Execution Guide

- Open a terminal in the project root folder and run “javac -d bin src/common/.java src/utils/.java src/server/.java src/client/.java”
- Start the Server: The server requires a port number argument (e.g., 8080): “java -cp bin server.ServerApp 8080”
- Start the Client: Open a new terminal. The client requires the host and port arguments: “java -cp bin client.ClientApp localhost 8080”