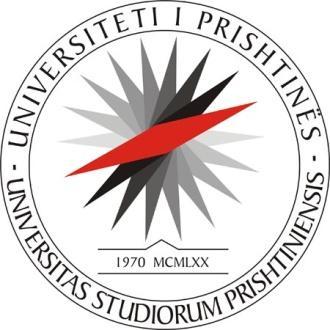
**University of Prishtina “Hasan Prishtina”**

Faculty of Electrical and Computer Engineering

10000 - Pristina, Kosovo



Water Quality Monitoring System

**Professor: Students:**

Besmir Sejdiu Rina Shabani

Albiona Vukaj

Gentrit Ibishi

**Table of content**

[1.](#_30j0zll) Introduction 4

[1.1 Purpose and Objectives 5](#_3znysh7)

[2.](#_2et92p0) System Architecture 5

[2.1 Components 6](#_3dy6vkm)

[2.1.1 Sensors 6](#_1t3h5sf)

[2.1.2 Tinkercad 6](#_4d34og8)

[2.1.3 Apache Kafka 7](#_17dp8vu)

[2.1.4 Apache Spark Streaming 8](#_3rdcrjn)

[2.1.5 Apache Cassandra 8](#_26in1rg)

[2.1.6 Java Spring Boot 9](#_lnxbz9)

[2.1.7 Angular 9](#_35nkun2)

[2.2 Data Flow and Component Interactions 10](#_1ksv4uv)

[3.](#_44sinio) Data Schema 10

[3.1 Sensor Data 10](#_2jxsxqh)

[3.2 Constrains 11](#_z337ya)

[4.](#_3j2qqm3) Integration and Implementation 12

[4.1 Integration with Apache Kafka 12](#_1y810tw)

[4.2 Processing with Apache Spark Streaming 13](#_4i7ojhp)

[4.3 Data Storage with Apache Cassandra 14](#_2xcytpi)

[5.](#_1ci93xb) Configuration and Setup 15

[5.1 Docker Configuration 15](#_3whwml4)

[5.2 Docker Commands 17](#_49x2ik5)

[5.2 Dependencies 21](#_3fwokq0)

[6.](#_1v1yuxt) Frontend Visualization of the Water Quality Monitoring System 22

[6.1 Technical Implementation 26](#_46r0co2)

[7.](#_111kx3o) Artificial Intelligence (AI) Implementation 26

[7.1 Data Collection and Preparation 27](#_3l18frh)

[7.3 Model Training 27](#_4k668n3)

[7.4 Real-time Prediction: 28](#_2zbgiuw)

[7.5 Self-Learning and Model Adaptation 29](#_3ygebqi)

[7.6 AI Workflow in the Spark Streaming System: 29](#_sqyw64)

[8.](#_3cqmetx) Alarm System 30

[8.1 Anomaly Identification 30](#_1rvwp1q)

[8.2 Recording and Storing Anomalies 30](#_4bvk7pj)

[8.3 Real-time Alarm Generation 31](#_2r0uhxc)

[8.4 Sending Informative Emails 31](#_1664s55)

[9.](#_34g0dwd) Performance Analysis and Optimization 33

[Conclusion 34](#_43ky6rz)

***Table of Figure***

[Figure 1 Tinkercad Sensor Simulation 6](#_2s8eyo1)

[Figure 2 Docker 16](#_2p2csry)

[Figure 3 cqlsh 19](#_41mghml)

[Figure 4 Cassandra 19](#_2grqrue)

[Figure 5 Spark Streaming 19](#_vx1227)

[Figure 6. Dashboard 22](#_19c6y18)

[Figure 7 Locations in Map 23](#_3tbugp1)

[Figure 8 Statistics based on location 23](#_28h4qwu)

[Figure 9 Parameters data over time 24](#_nmf14n)

[Figure 10 Dashboard Alert 24](#_37m2jsg)

[Figure 11 AI Risk Alerts 25](#_1mrcu09)

[Figure 12 Received emails 30](#_3q5sasy)

[Figure 13 Email for "System Started" 31](#_25b2l0r)

[Figure 14 Email for "Detection Anomaly" 31](#_kgcv8k)

[Figure 15 Performance Analysis 33](#_1jlao46)

**Abstract**

The rapid degradation of water resources due to pollution and climate change has heightened the need for effective water quality monitoring systems. This project presents an IoT-based Water Quality Monitoring System designed to provide real-time data collection, transmission, processing, and visualization of critical water parameters such as pH, temperature, turbidity, and flow rates. The system leverages state-of-the-art technologies including Apache Kafka for reliable data transmission, Apache Spark Streaming for real-time data processing, and Apache Cassandra for scalable data storage. Angular-based user interface facilitates intuitive visualization and interaction with the processed data. The primary objective of this project is to establish a robust and scalable infrastructure for continuous monitoring of water quality across multiple locations. The system is capable of detecting anomalies in water parameters, triggering alerts for potential risks, and providing historical data analysis for trend identification. In the absence of physical sensors, data simulation is implemented to replicate real-world conditions. The successful deployment of this system is expected to significantly enhance water quality monitoring capabilities, offering a real-time view of water conditions and enabling proactive measures to safeguard water resources. This project not only demonstrates the potential of IoT technologies in environmental monitoring but also paves the way for integrating artificial intelligence for predictive analytics, further contributing to the development of smart and sustainable ecosystems.

# Introduction

Water quality is a critical factor for the health and well-being of ecosystems, as well as for human consumption and recreational activities. With increasing pollution and climate change, monitoring water quality has become more important than ever to ensure the safety and availability of this vital resource. Traditional water monitoring methods, which often rely on manual sampling and laboratory analysis, are time-consuming and lack the capability to provide real-time insights. This has created a demand for more efficient, automated, and continuous monitoring systems.

The Water Quality Monitoring System presented in this project addresses these challenges by leveraging technologies such as Apache Kafka, Apache Spark Streaming, and Cassandra for data processing and storage, combined with a web interface built with Angular for data visualization. The system is deployed across four key water bodies in Kosovo: Liqeni i Ujmanit, Liqeni i Badovcit, Liqeni i Batllaves, and Liqeni i Radoniqit. It continuously monitors crucial water quality parameters such as pH, turbidity, temperature, and flow, transmitting the data from various sensors to a Kafka producer. From there, the data is ingested by Spark Streaming for real-time processing and anomaly detection before being stored in a Cassandra database.

The system's dashboard, designed in Angular, provides a comprehensive and user-friendly interface for visualizing water quality data. It features an interactive map where each monitored location is represented by a color-coded icon: green for drinkable water, red for non-drinkable, and yellow for detected anomalies. By clicking on any location, users can view detailed, real-time information on the water quality parameters specific to that site.

## 1.1 Purpose and Objectives

The primary purpose of this project is to develop an automated and scalable solution for real-time water quality monitoring, leveraging IoT and machine learning technologies. The objectives of this project are:

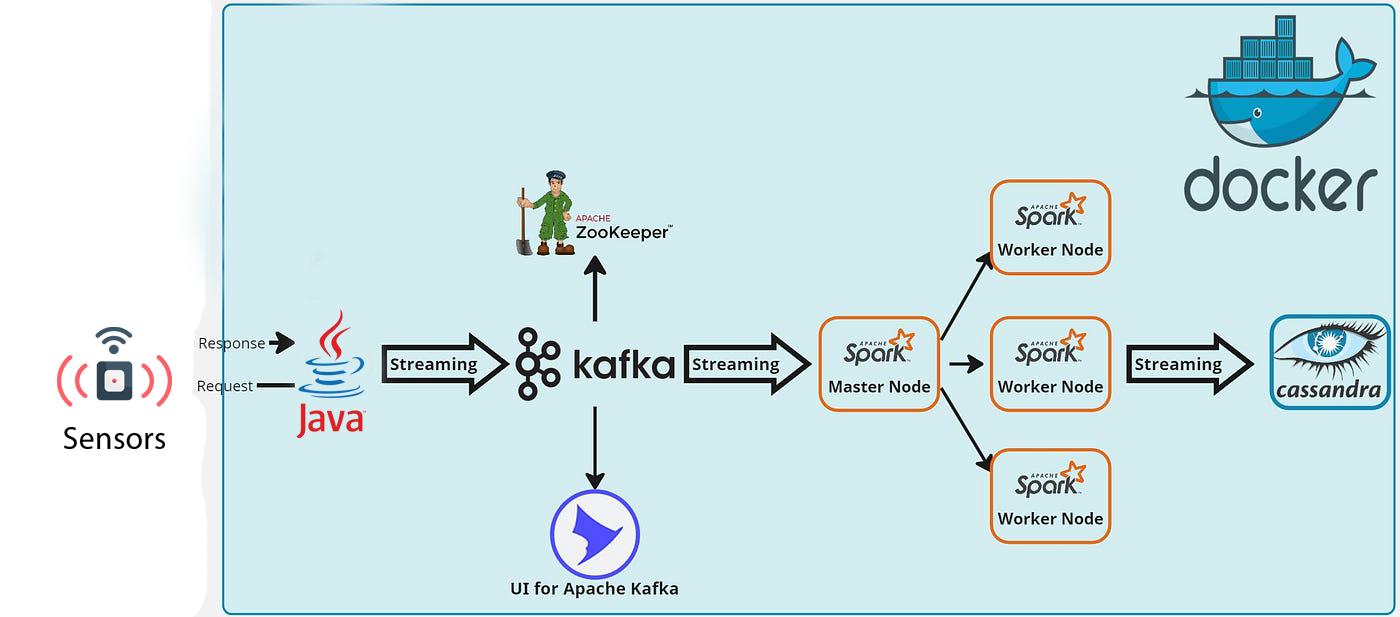
1. **Real-time Monitoring and Data Processing:** The system aims to provide continuous, real-time monitoring of water quality parameters across multiple locations. Using a Kafka-based data pipeline and Spark Streaming, it processes sensor data in real-time to detect any significant changes or anomalies.
2. **Accurate Classification and Anomaly Detection:** To classify water quality as drinkable, non-drinkable, or anomalous, the system employs a Logistic Regression model. This model, initially trained on a historical dataset of over 100,000 samples, continuously learns from new data to improve its accuracy and reliability. Anomalies are detected using a Z-score method, which helps in identifying sudden and unexpected changes in water quality.
3. **Comprehensive Data Visualization:** The system's dashboard is designed to provide a clear and intuitive overview of water quality status across all monitored locations. It includes a pie chart summarizing the proportion of data classified as drinkable, non-drinkable, and anomalous, and a map that provides quick access to real-time data for each location.
4. **Historical Data Analysis and Trend Identification:** The statistics module enables users to filter and analyze historical data, visualizing trends over time for each parameter and location. This helps in understanding long-term changes in water quality and identifying potential areas of concern.
5. **Automated Alerting and Response Mechanism:** The AI risk alert module enhances the system's functionality by detecting anomalies and suggesting mitigation actions. This module not only triggers visual alerts on the dashboard but also sends automatic email notifications to relevant authorities, ensuring a prompt response to potential water quality issues.

# System Architecture

The Water Quality Monitoring System is designed as an integrated solution for real-time monitoring, processing, and visualization of water quality parameters. It consists of several interconnected components that work seamlessly to provide accurate and timely information about the water conditions at four key locations: Liqeni Ujmanit, Liqeni Badovcit, Liqeni Batllaves, and Liqeni Radoniqit.

The architecture is structured around a central data pipeline that begins with data collection from sensors deployed at each location. This data is transmitted to a centralized server using Apache Kafka, where it is processed in real-time by Apache Spark Streaming. Processed data is then stored in Apache Cassandra for scalable, low-latency storage. A frontend dashboard built with Angular provides users with an interactive interface to view real-time data, historical trends, and AI-driven risk alerts.

The architecture also includes an AI module that continuously learns from the collected data, enhancing the accuracy of anomaly detection and classification.



*Figure 1 System Architecture*

## 2.1 Components

### 2.1.1 Sensors

The system utilizes various sensors to measure key water quality parameters such as pH, temperature, and turbidity. These sensors are deployed at each of the four monitoring locations and are configured to collect data at regular intervals. The data collected includes:

* **pH Sensor:** Measures the acidity or alkalinity of the water, which is crucial for assessing the suitability of the water for drinking and ecological health.
* **Temperature Sensor:** Monitors the water temperature, an important factor influencing the metabolic rates of aquatic organisms and the solubility of gases.
* **Turbidity Sensor:** Measures the clarity of the water, indicating the presence of suspended particles and potential pollutants.
* **Flow Sensor:** Measures the rate and volume of water movement in a system, providing essential data for water management and distribution. It helps in detecting leaks, monitoring usage, and understanding the dynamics of water flow within the system. Accurate flow measurements are crucial for calculating discharge rates and ensuring the efficiency of water treatment processes.

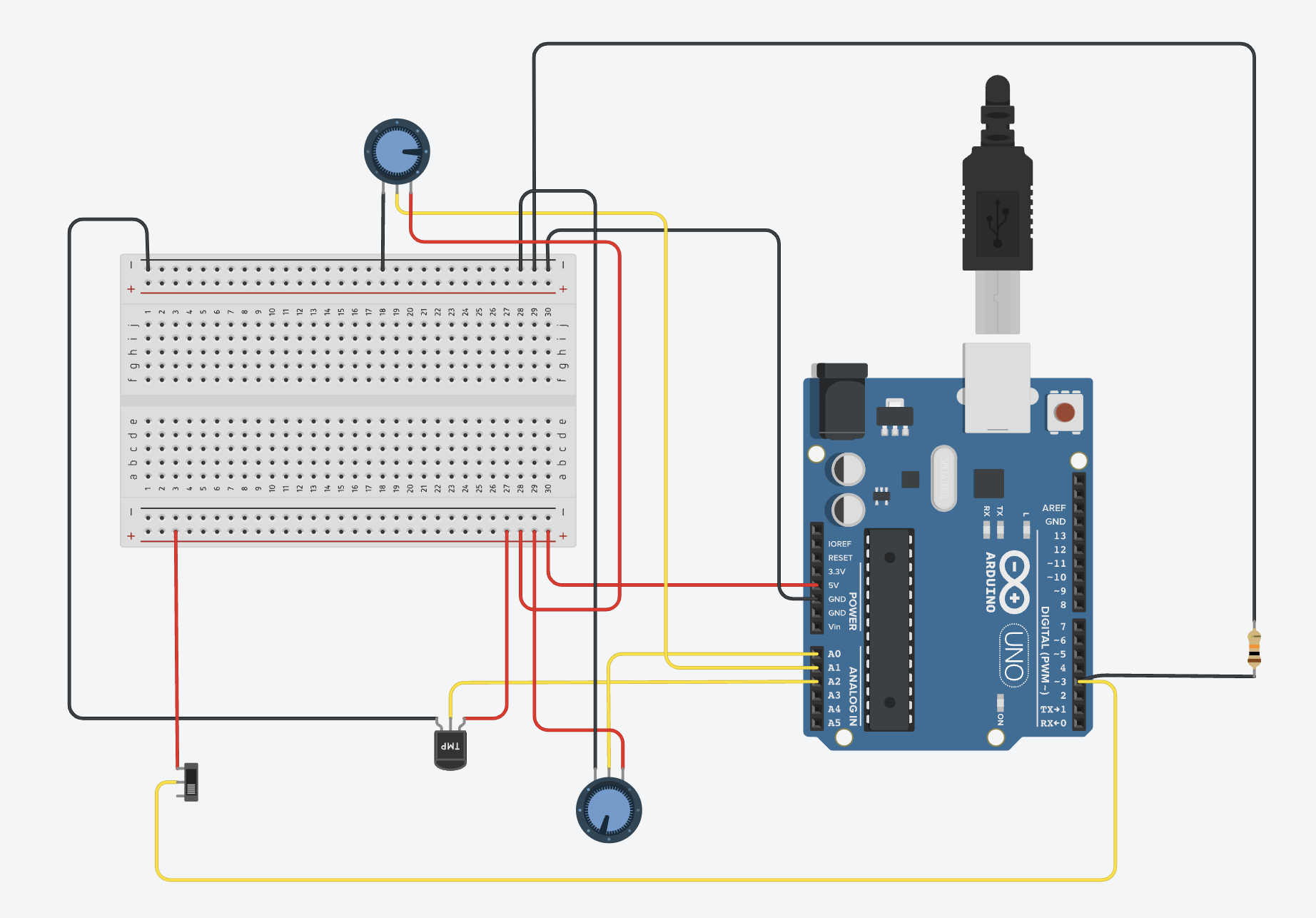
These sensors are connected to a local gateway that aggregates the data and prepares it for transmission to the central system.

### 2.1.2 Tinkercad

The Tinkercad setup represents a simulation for water quality monitoring system using an Arduino Uno board. The setup includes connections for various sensors such as pH, turbidity, flow, and temperature sensors, integrated as follows:

1. **Arduino Uno Board**: The central microcontroller used to interface with the sensors and handle data processing and communication.
2. **Breadboard**: Used for wiring and connecting different components without soldering. It facilitates easy modifications and testing of the circuit.
3. **pH Sensor**: This sensor measures the acidity or alkalinity of water. It is connected to the analog pin of the Arduino to read the pH value.
4. **Turbidity Sensor**: It measures the cloudiness or haziness of water, indicating the presence of suspended solids. The sensor's analog output is connected to another analog pin on the Arduino.
5. **Flow Sensor**: Measures the flow rate of water, indicating the amount of water passing through the system per unit time. It is connected to a digital pin on the Arduino to count pulses, which can be translated into flow rate.
6. **Temperature Sensor**: Reads the water temperature. It is connected to an analog pin for temperature data acquisition.
7. **Power Connections**: The breadboard and sensors are powered through the Arduino board using the 5V and GND pins.
8. **USB Connection**: The Arduino board is powered through a USB connection, which also allows for communication with a computer for programming and data logging.

This setup enables the simulation of a water quality monitoring system, allowing for the measurement and monitoring of key water quality parameters using Arduino.

****

*Figure 2 Tinkercad Sensor Simulation*

### 2.1.3 Apache Kafka

Apache Kafkais a distributed streaming platform designed for building real-time data pipelines and applications. It acts as a high-throughput, fault-tolerant message broker that enables the storage, management, and processing of streams of records in real-time. Kafka organizes data into topics, and producers send data to these topics while consumers read data from them. It is widely used for use cases like real-time analytics, event tracking, log aggregation, and data integration across various systems, making it a cornerstone technology for scalable and reliable data streaming.

**Key Kafka Components:**

* **Producers:** Each sensor gateway is configured as a Kafka producer, sending real-time data to Kafka topics.
* **Topics:** Separate topics are created for each parameter and location, enabling organized and parallel data processing.
* **Consumers:** Apache Spark Streaming acts as a consumer, subscribing to these topics to process the incoming data.

### 2.1.4 Apache Spark Streaming

**Apache Spark Streaming** is a scalable and fault-tolerant stream processing system built on Apache Spark. It enables real-time data processing of continuous data streams from sources like Apache Kafka, file systems, and more. Spark Streaming processes data in small, manageable batches, allowing for high-throughput and low-latency computations. It supports complex event processing, data transformations, and machine learning algorithms on streaming data. This makes it ideal for use cases such as real-time analytics, monitoring, and live data processing in various industries.

**Data Processing:** Apache Spark Streaming is employed for real-time data processing. As the data is streamed from Kafka, Spark processes it to perform several key functions:

**Data Validation:** Ensures the integrity and accuracy of incoming data by filtering out any anomalous readings that fall outside predefined thresholds.

**Anomaly Detection:** Uses a logistic regression model to classify the water quality and detect any deviations from normal patterns.

**Aggregation and Analysis:** Aggregates data over time windows to provide meaningful insights into water quality trends and metrics.

This real-time processing capability enables the system to respond promptly to changes in water quality, triggering alerts and updating the visualization dashboard.

### 2.1.5 Apache Cassandra

**Apache Cassandra:** A highly scalable, distributed NoSQL database designed to handle large amounts of data across many nodes with high availability and no single point of failure. Cassandra is ideal for real-time data storage and retrieval, offering a flexible schema and support for horizontal scaling. It is commonly used for applications that require high write throughput and the ability to handle massive volumes of data, making it a perfect choice for storing time-series data and the processed results in water quality monitoring systems.

**Data Storage:** The processed data is stored in Apache Cassandra, a NoSQL database known for its high availability, scalability, and low-latency performance. Cassandra is well-suited for handling the large volume of data generated by the sensors and supports efficient querying for both real-time and historical data analysis.

**Key Features of the Storage System:**

* **Scalable Architecture:** Allows for horizontal scaling, accommodating the increasing volume of sensor data as the system grows.
* **High Availability:** Ensures data is always accessible, even in the event of node failures.
* **Time-Series Data Management:** Supports efficient storage and retrieval of time-series data, which is critical for tracking changes in water quality over time.

### 2.1.6 Java Spring Boot

Java Spring Boot is an open-source framework designed to simplify the development of Java applications, particularly for building enterprise-level web applications and microservices. It is built on the Spring framework and provides a set of conventions and tools that enable developers to create stand-alone, production-ready applications with minimal configuration. Key features include:

* **Auto-Configuration**: Spring Boot automatically configures application settings based on the included libraries, reducing the need for boilerplate code.
* **Embedded Servers**: It allows developers to run applications on embedded servers like Tomcat simplifying deployment and testing.
* **Production-Ready Features**: Built-in features such as health checks, metrics, and externalized configuration make it easy to monitor and manage applications in production.
* **RESTful API Development**: Spring Boot facilitates the creation of RESTful APIs, making it an excellent choice for building web services and microservices architectures.

Overall, Spring Boot accelerates the development process, promotes best practices, and enhances productivity, making it a popular choice among Java developers for building scalable and maintainable applications.

### 2.1.7 Angular

**Angular**: A powerful, open-source web application framework developed by Google for building dynamic, single-page applications (SPAs). Angular provides a comprehensive toolkit, including two-way data binding, dependency injection, and a modular architecture, making it easy to create responsive and interactive user interfaces. It is commonly used for developing complex front-end applications with real-time data visualization and user-friendly dashboards, such as those in water quality monitoring systems.

**Visualization:** The system’s frontend is developed using Angular, providing an interactive dashboard that allows users to monitor water quality in real-time. Key features of the dashboard include:

**Map Interface:** Displays each of the four monitoring locations with color-coded markers that indicate the current water quality status.

**Real-Time Data Display:** Allows users to hover over each location to view the latest data readings for pH, turbidity, and temperature.

**Alert System:** Automatically triggers visual alerts on the dashboard and sends email notifications when anomalies are detected.

**Historical Data Analysis:** Provides graphs and charts that show historical trends in water quality parameters, allowing users to identify patterns and assess the effectiveness of interventions.

This robust architecture ensures that the Water Quality Monitoring System is capable of providing reliable, real-time insights into the status of water resources, supporting proactive management and timely response to potential water quality issues.

## 2.2 Data Flow and Component Interactions

**Data Collection:** Sensors measure water quality parameters and send data to the local gateway.

**Data Transmission:** Data from the gateway is published to Apache Kafka topics, where each topic corresponds to a specific location or parameter.

**Data Processing:** Apache Spark Streaming consumes data from Kafka, processes it in real-time to detect anomalies, and aggregates it for further analysis.

**Data Storage:** The processed data is stored in Apache Cassandra, ensuring high availability and scalability for real-time and historical data retrieval.

**Visualization and Alerts:** The data is visualized on the Angular-based dashboard, and any detected anomalies trigger alerts, which are displayed on the dashboard and sent via email notifications.

# Data Schema

The data schema for the Water Quality Monitoring system is designed to efficiently store real-time sensor data related to water quality parameters. This schema is implemented in Apache Cassandra, selected for its capability to handle high-velocity data and provide scalability and reliability.

## 3.1 Sensor Data

The readings table is the core table that captures and stores all real-time data collected from various water quality sensors. This table includes fields for storing sensor measurements, geographical coordinates, and metadata related to each reading.

CREATE TABLE IF NOT EXISTS sensor\_data.readings (

id UUID PRIMARY KEY,

temperature TEXT,

ph DOUBLE,

flow INT,

turbidity DOUBLE,

latitude DOUBLE,

longitude DOUBLE,

address TEXT,

timestamp TIMESTAMP,

prediction INT,

time\_execution BIGINT

);

**Description of Columns:**

* **id:** A unique identifier (UUID) for each sensor reading, ensuring that each entry is distinct.
* **temperature:** Stores the water temperature as a text field, indicating the temperature at the time of reading.
* **ph:** Represents the pH level of the water, which measures the acidity or alkalinity and is crucial for assessing water quality.
* **flow:** Indicates the flow rate of the water, which can be used to monitor the water movement and distribution.
* **turbidity:** Measures the turbidity or cloudiness of the water, which can be an indicator of contamination or water quality issues.
* **latitude & longitude:** Geographic coordinates specifying the exact location of the sensor, which helps in mapping and tracking the data source.
* **address:** A textual description of the sensor location, providing human-readable information about the monitoring site.
* **timestamp:** Records the date and time when the sensor reading was taken, allowing for chronological analysis of data.
* **prediction:** Represents the predicted quality status of the water based on the AI model. It uses integer values to indicate different states:
  + 0 = Not Drinkable
  + 1 = Drinkable
  + 2 = Anomaly
* **time\_execution:** Stores the time taken to process the sensor data, measured in milliseconds, which helps in performance monitoring of the data processing pipeline.

## 3.2 Constrains

**Water Quality Constraints for Drinkable and Not Drinkable Classification**

The water quality monitoring system uses specific constraints to classify water as either "Drinkable" or "Not Drinkable" based on the readings from various sensors. The constraints are defined for four key water quality parameters: flow, temperature, turbidity, and pH. These parameters are monitored in real-time to assess the overall quality of the water.

* **Flow Rate Constraints**
* **Drinkable**:
  + If the flow rate is either 0 or 1, the water is considered drinkable.
* **Not Drinkable**:
  + If the flow rate is 2, the water is not considered drinkable.
* **Temperature Constraints**
* **Drinkable**:
  + If the temperature is within the range of 20°C to 25°C, the water is considered drinkable.
* **Not Drinkable**:
  + If the temperature is less than or equal to 20°C or greater than or equal to 25°C, specifically in the range of -10°C to 20°C or 25°C to 100°C, the water is considered not drinkable.
* **Turbidity Constraints**
* **Drinkable**:
  + If the turbidity value is between 0 and 2.4 NTU (Nephelometric Turbidity Units), the water is considered drinkable.
* **Not Drinkable**:
  + If the turbidity value is greater than 2.4 NTU and up to 1000 NTU, the water is considered not drinkable. High turbidity indicates the presence of suspended particles, making the water unsafe for drinking.
* **pH Constraints**
* **Drinkable**:
  + If the pH level is between 6.5 and 8.5, the water is considered drinkable. This range indicates a balanced pH level that is safe for consumption.
* **Not Drinkable**:
  + If the pH level is below 6.5, the water is considered too acidic and not drinkable.
  + If the pH level is above 8.5, the water is considered too alkaline and not drinkable.

# Integration and Implementation

In this project, several technologies and tools were integrated to build a comprehensive real-time water quality monitoring system. The core components include Apache Kafka, Apache Spark Streaming, and Apache Cassandra. Each component plays a vital role in ensuring efficient data collection, processing, and storage.

## 4.1 Integration with Apache Kafka

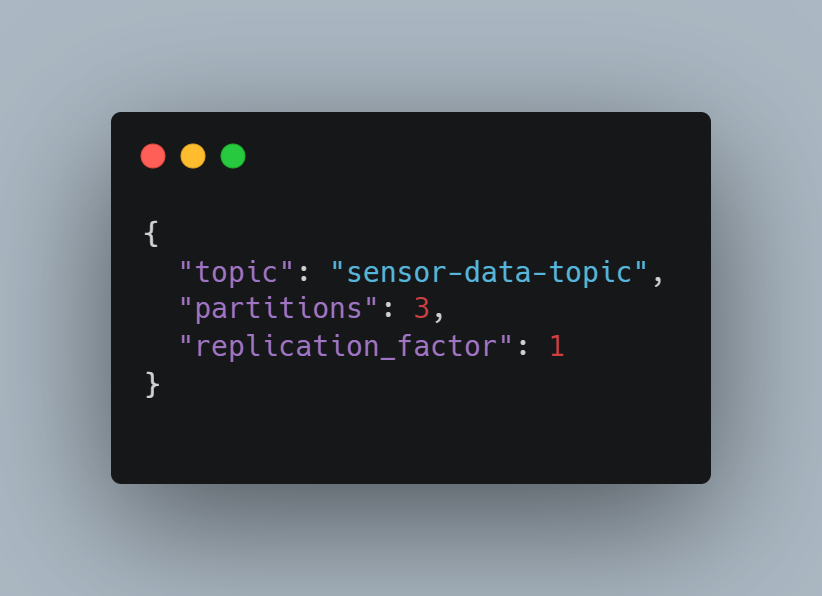
Apache Kafka serves as the backbone for real-time data ingestion and streaming. It is a distributed streaming platform that handles high-throughput, fault-tolerant, and scalable data streams. The Kafka producer sends real-time data from various water quality sensors to Kafka topics, ensuring reliable and continuous data transmission.

Purpose in the Water Quality Monitoring System: Kafka is used to transmit sensor data from various sources to the data processing layer in real-time. It ensures that data is streamed continuously and efficiently to the Spark Streaming system without loss or delay.

**Kafka Configuration:**

1. Set up Kafka Brokers:
   * Kafka brokers are configured to manage and store data streams. Multiple brokers ensure high availability and fault tolerance.
2. Configure Topics for Data Ingestion:
   * Data is published to specific Kafka topics that represent the data streams from water quality sensors. These topics are configured to handle the high volume of data and to distribute it efficiently for parallel processing.

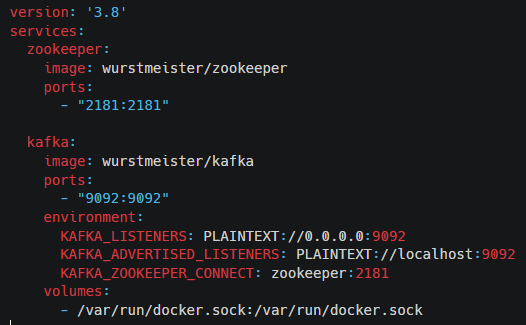
**Kafka topic configuration:**



* **topic**: The name of the topic (sensor -topic) representing the stream of messages from the sensors.
* **partitions**: The number of partitions allows Kafka to scale horizontally by distributing data across multiple brokers.
* **replication\_factor:** The number of copies of the data kept across different brokers to ensure redundancy and fault tolerance.

**Data Transmission:** The Kafka producer sends sensor data to the configured topics using the producer API. This ensures reliable and real-time transmission of water quality data from each sensor node to the central processing system.

Running Kafka: Kafka is run locally using Docker Compose with the following configuration:



This configuration ensures that sensor data is transmitted reliably and efficiently, supporting real-time data processing and analysis.

## 4.2 Processing with Apache Spark Streaming

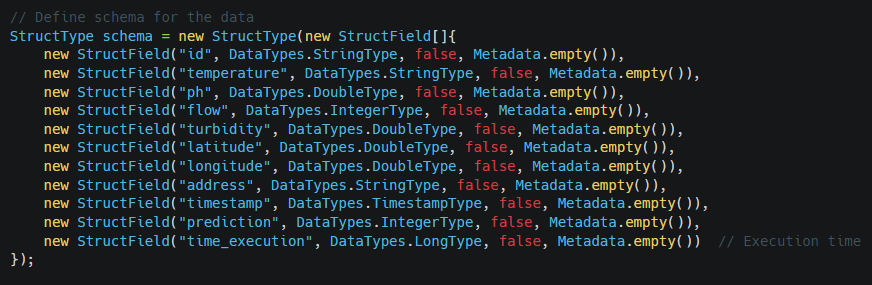
Apache Spark Streaming is used to process the real-time data streams received from Kafka. It provides a scalable and fault-tolerant framework for real-time data processing and analysis.

Purpose in the Water Quality Monitoring System: Spark Streaming processes real-time sensor data, allowing for immediate analysis and detection of water quality issues. It enables the system to perform real-time classification and anomaly detection, providing actionable insights.

**Spark Streaming Configuration:**

1. Set up a Spark Streaming Context:
   * A Spark Streaming context is configured with the necessary settings to read from Kafka and write to Cassandra.
2. Real-Time Data Processing:
   * The system reads data from the Kafka topic, processes it in real-time using Spark Streaming, and performs necessary transformations such as data parsing, anomaly detection, and classification.

Spark Streaming processing:



1. Writing Processed Data to Cassandra:
   * After processing the data, the structured data is written to a Cassandra table for efficient storage and retrieval.

By using Apache Spark Streaming, the system can process large volumes of data in real-time, ensuring that information about water quality is always up-to-date.

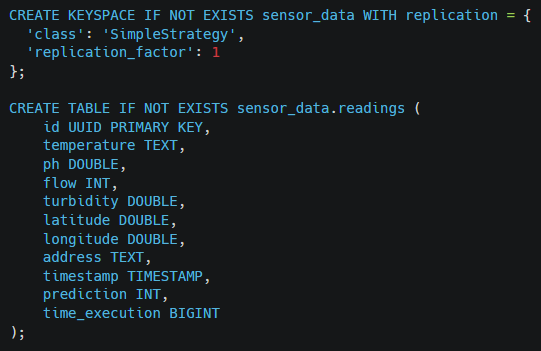
## 4.3 Data Storage with Apache Cassandra

Apache Cassandra is used to store the processed sensor data, providing a robust backend for real-time data analysis. Cassandra is a highly scalable NoSQL database designed to handle large volumes of data with high availability and fault tolerance.

Purpose in the Water Quality Monitoring System: Cassandra stores both the raw sensor data and the processed results. It ensures that the data is available for real-time analysis and historical review, providing a reliable storage solution for the water quality monitoring system.

**Cassandra Configuration:**

1. Define Keyspace and Table Structure:
   * The keyspace and table for storing sensor data are defined with appropriate data types and configurations to handle the real-time ingestion of data.



Integration with Spark Streaming: The Spark Cassandra connector is used to enable Spark jobs to read from and write to Cassandra efficiently. This integration ensures that real-time data processing results are immediately available in Cassandra for further analysis and reporting.

By integrating Apache Cassandra, the system ensures that all collected and processed data is stored in a highly available and scalable manner, enabling real-time data retrieval and analysis. This makes Cassandra an ideal choice for the backend of the water quality monitoring project.

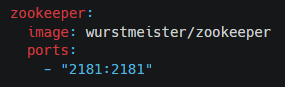
# Configuration and Setup

In this project, we configured a multi-container environment using Docker Compose to facilitate real-time data processing and analysis. The setup includes four interconnected services: Zookeeper, Kafka, Cassandra, and Spark, each serving a specific role in the data pipeline. Below is a detailed description of the configuration:

## 5.1 Docker Configuration

#### **1. Zookeeper Service**

For the coordination and management of the Kafka broker, I deployed a Zookeeper service using the wurstmeister/zookeeper Docker image. Zookeeper is crucial for maintaining metadata, managing service discovery, and ensuring the high availability of the Kafka cluster. I exposed port 2181 to the host machine to allow communication between Zookeeper and the Kafka service.

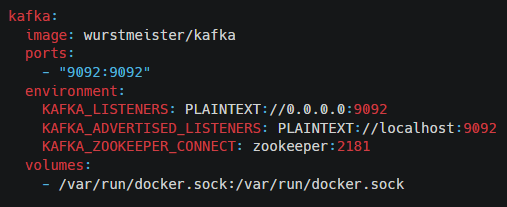


#### **2. Kafka Service**

We set up a Kafka broker using the wurstmeister/kafka Docker image. This service handles real-time data streams and acts as the backbone of the data pipeline. I configured it with several environment variables to integrate it seamlessly with Zookeeper and manage its advertised listeners:

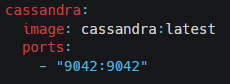
* KAFKA\_LISTENERS: Configured to listen on all network interfaces (0.0.0.0:9092).
* KAFKA\_ADVERTISED\_LISTENERS: Set to localhost:9092 to ensure that clients can connect to Kafka from outside the container.
* KAFKA\_ZOOKEEPER\_CONNECT: Links Kafka to the Zookeeper service (zookeeper:2181) for coordination.

Additionally, I mounted the Docker socket (/var/run/docker.sock) to the container to allow Kafka to detect its environment and adjust configurations dynamically.



#### **3. Cassandra Service**

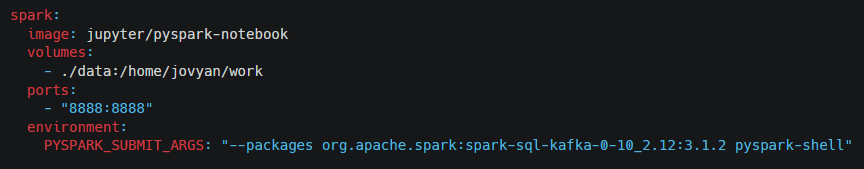
For data persistence, I chose Cassandra as the NoSQL database, deploying it using the latest cassandra Docker image. This service stores processed data, enabling efficient and scalable data retrieval. I exposed port 9042 to allow access to Cassandra’s CQL interface from the host machine, facilitating data queries and analysis.



#### **4. Spark Service**

To process and analyze the data streams, I set up a Spark service using the jupyter/pyspark-notebook image, which includes a Jupyter notebook environment pre-configured with PySpark. This setup provides a user-friendly interface for data exploration and model development.

* **Volume Mapping:** I mapped the local ./data directory to the container’s /home/jovyan/work directory. This allows me to seamlessly share data files between the host and the Spark container, facilitating data analysis and experimentation.
* **Port Configuration:** I exposed port 8888 to access the Jupyter notebook interface from the host machine.
* **Environment Configuration:** I configured the PYSPARK\_SUBMIT\_ARGS environment variable to include the Kafka integration package (spark-sql-kafka-0-10\_2.12:3.1.2), enabling Spark to read from and write to Kafka topics, thus integrating it into the real-time data pipeline.



## 5.2 Docker Commands

* **Starting All Defined Services:**
* To launch all services as specified in the docker-compose.yml file, use the command:

**

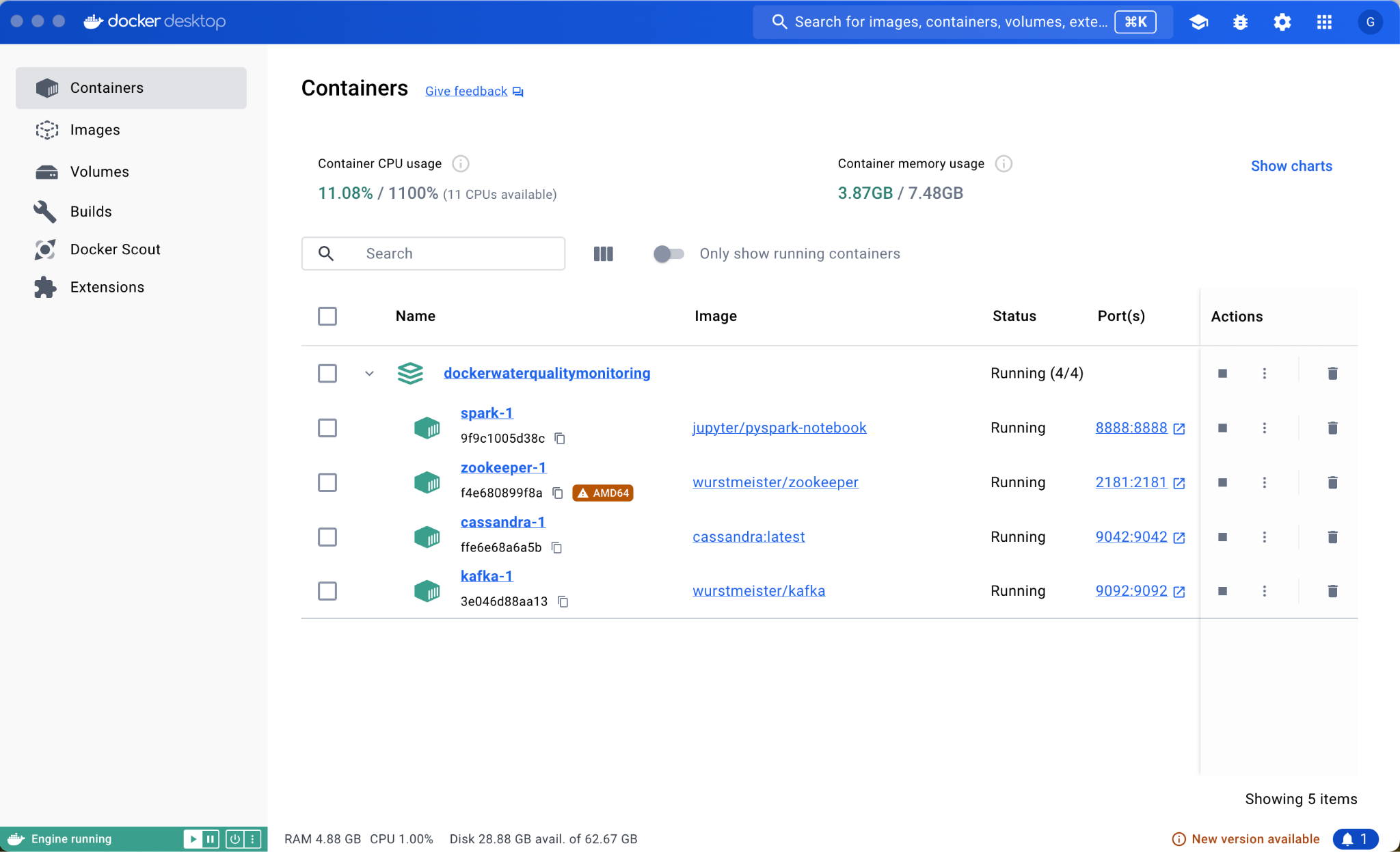
* This will bring up all the services in the configuration, rebuilding any images as necessary.

**Accessing the Cassandra Shell:**

* To interact with the Cassandra database directly, you can enter the shell environment using the following command:



* This will open the Cassandra Query Language (CQL) shell, allowing you to run various CQL commands.



*Figure 3 Docker*

**Real-time Data Stream Ingestion:**

Spark Streaming serves as the central processing engine that consumes real-time data from a Kafka producer. The data streams represent sensor readings for various water quality parameters such as temperature, pH, turbidity, and flow rate. The data ingestion process is facilitated through a Kafka direct stream, which provides high throughput and fault tolerance.

* **Kafka Direct Stream Configuration:**
  + A Kafka Direct Stream is created using the KafkaUtils.createDirectStream method, which directly connects to the Kafka broker and fetches messages without the need for an intermediate receiver.
  + The configuration includes parameters such as bootstrap.servers, key.deserializer, and value.deserializer, along with consumer group settings and auto-offset management. This ensures that data is consumed consistently and efficiently from the specified Kafka topic, “SensorTopic”.

**Data Schema Definition:**

Once the data is ingested, it needs to be structured to facilitate further processing. The system defines a schema using Spark's StructType and StructField classes, which specify the expected format of the data.

* **Schema Fields:**
  + id: A unique identifier for each sensor reading (StringType).
  + temperature: The temperature reading from the sensor (StringType).
  + ph: The pH level of the water, representing its acidity or alkalinity (DoubleType).
  + flow: The flow rate of water, indicating the volume of water passing through a point (IntegerType).
  + turbidity: A measure of water clarity, with higher values indicating cloudier water (DoubleType).
  + latitude and longitude: Geographical coordinates of the sensor location (DoubleType).
  + address: A textual description of the sensor location (StringType).
  + timestamp: The time at which the sensor data was recorded (TimestampType).
  + prediction: A field indicating the result of the AI model’s prediction (IntegerType).
  + time\_execution: The total time taken to process a batch (LongType).

The structured schema allows the system to create DataFrames, which are then used for structured and optimized data processing.

**Micro-batch Processing:**

Spark Streaming operates in micro-batch mode, where data is processed in small intervals, known as micro-batches. This approach combines the advantages of both real-time streaming and batch processing.

* **Batch Interval:**
  + The batch interval is set to 5 seconds (Durations.seconds(5)), meaning that data is collected and processed every 5 seconds. This low-latency processing ensures that the system remains responsive and up-to-date with the latest sensor readings.
  + The batch interval can be adjusted to optimize performance based on the system's requirements and workload characteristics. Shorter intervals provide more real-time updates but may increase processing overhead, while longer intervals reduce the processing frequency but can handle larger data volumes per batch.
* **Batch Processing Workflow:**
  + For each micro-batch, the system records the start time (batchStartTime) and processes all records within that batch.
  + The data is then converted into a structured format (DataFrame) using the defined schema, enabling the use of SQL-like queries and transformations.

**Data Transformation:**

After ingestion and structuring, the data undergoes various transformations to prepare it for analysis and storage.

* **JSON Parsing:**
  + Each record is parsed from JSON format, extracting the necessary fields such as temperature, pH, flow, and turbidity. The extracted values are then converted to their respective data types.
  + Any malformed or incomplete records are filtered out to ensure data quality.
* **Computations and Aggregations:**
  + The system performs various computations such as counting the number of records processed in each batch (rdd.count()) and calculating the total execution time for the batch.
  + Additional transformations may include data normalization, scaling, or feature engineering to prepare the data for AI modeling and anomaly detection.

**Anomaly Detection:**

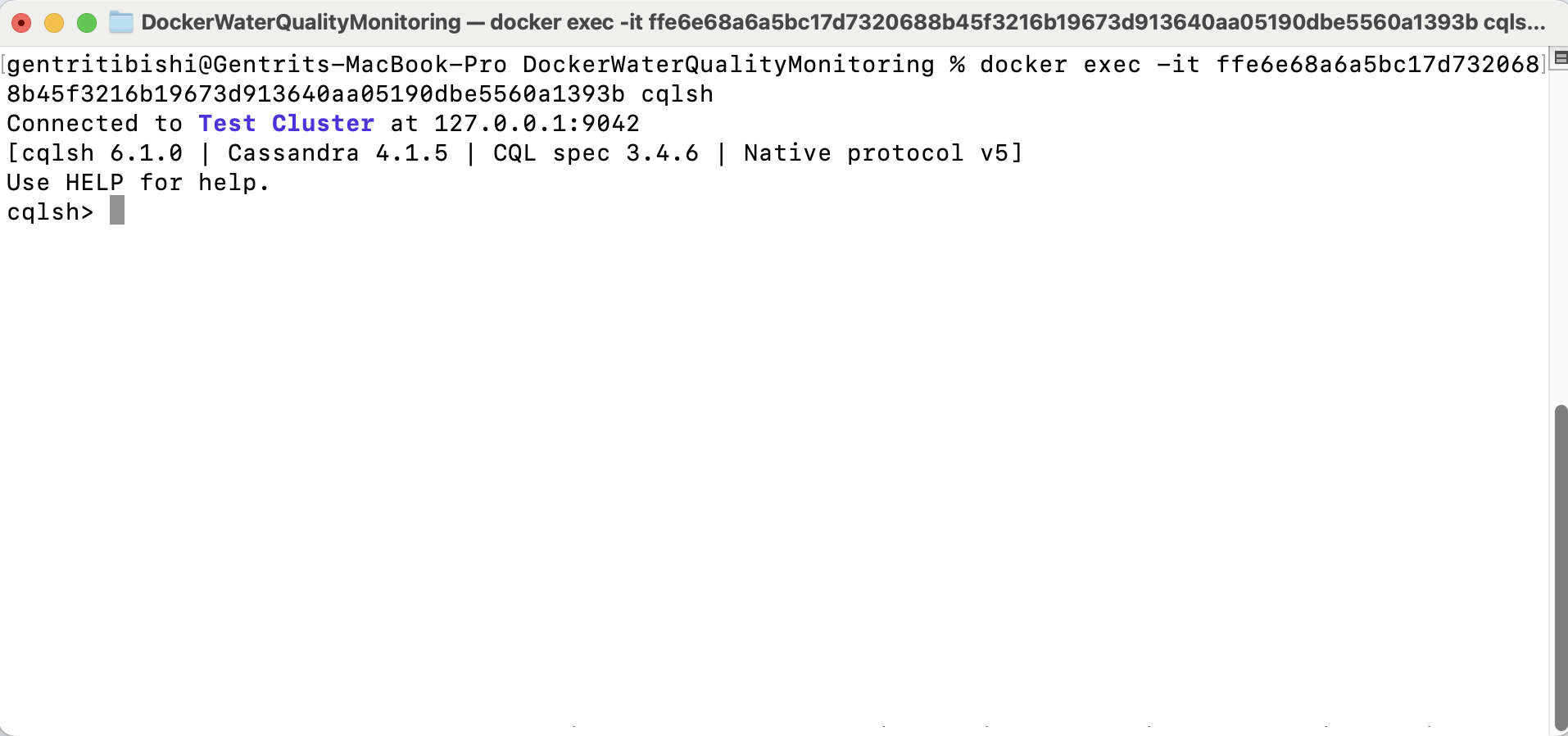
Anomaly detection is a critical component of the system, helping to identify unexpected or abnormal sensor readings that may indicate issues such as sensor malfunctions or environmental anomalies.

* **Z-score Calculation:**
  + The system calculates the Z-score for each sensor reading, which is a measure of how many standard deviations a value is from the mean.
  + If the absolute Z-score of a reading exceeds a predefined threshold (e.g., 3.0), it is flagged as an anomaly. This approach helps in identifying outliers that deviate significantly from normal behavior.
* **Triggering Alerts:**
  + When an anomaly is detected, the system triggers an alert, which can include visual indicators on the dashboard, email notifications, or other actions as defined by the system's alerting mechanism.
  + Anomalies are stored with additional metadata, including the timestamp and location, for further investigation and historical reference.

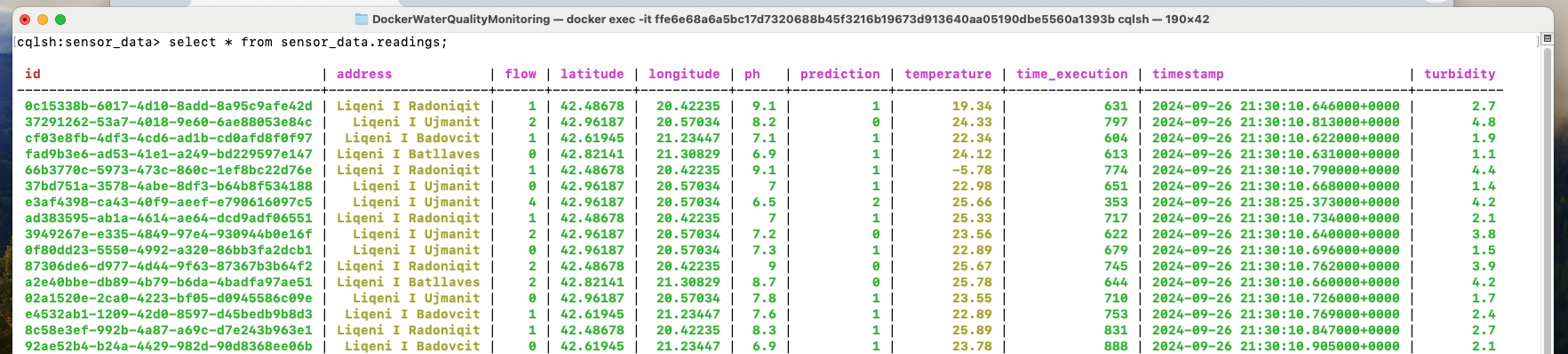
**Data Storage and Persistency:**

After processing and anomaly detection, the data is persisted in a Cassandra database. This allows for scalable and fault-tolerant storage of large volumes of time-series data.

* **Cassandra Integration:**
  + The processed DataFrame is saved to the Cassandra database using the save method with format("org.apache.spark.sql.cassandra").
  + The data is stored in the "sensor\_data" keyspace and the "readings" table, with the mode set to SaveMode.Append, ensuring that new records are added without overwriting existing data.



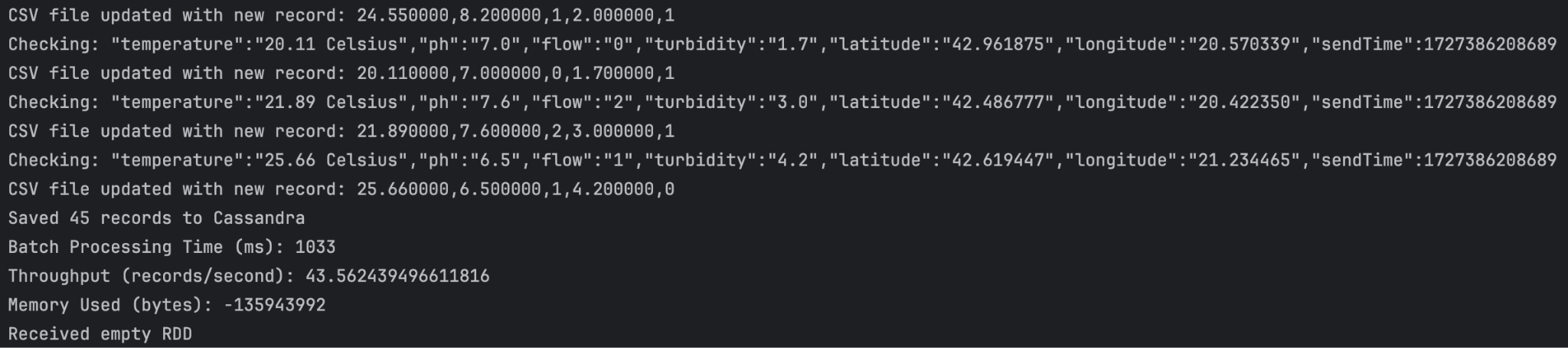
*Figure 4 cqlsh*



*Figure 5 Cassandra*

* **Historical Data Analysis:**
  + Storing processed data in Cassandra enables historical analysis, allowing the system to query past readings, detect trends, and improve the accuracy of predictive models over time.
  + The stored data can also be used for reporting, compliance, and system audits.

By employing this comprehensive data processing and aggregation workflow, the Spark Streaming system ensures that incoming sensor data is efficiently ingested, processed, and stored, providing a robust foundation for real-time monitoring and analysis of water quality.

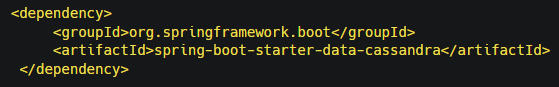


*Figure 6 Spark Streaming*

## 5.2 Dependencies

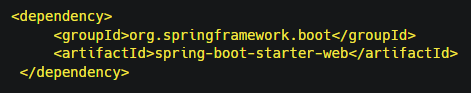
**1. Spring Boot Starter Data Cassandra**

Purpose: Provides support for integrating with the Cassandra NoSQL database, enabling data persistence and querying.



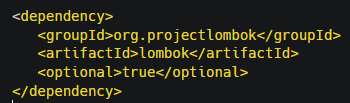
**2. Spring Boot Starter Web**

Purpose: Enables building web applications, including RESTful APIs using Spring MVC.



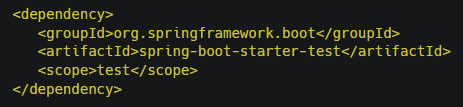
**3. Lombok**

Purpose: Reduces boilerplate code by generating getters, setters, constructors, and other utility methods.



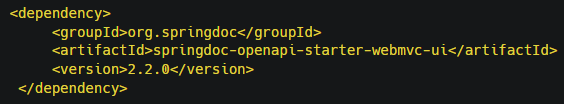
**4. Spring Boot Starter Test**

Purpose: Provides testing support with tools like JUnit, Hamcrest, and Mockito.



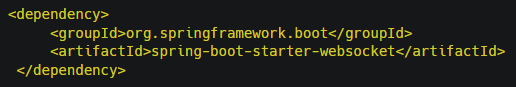
**5. SpringDoc OpenAPI**

Purpose: Integrates OpenAPI 3 with Spring Boot for generating API documentation and providing a Swagger UI.



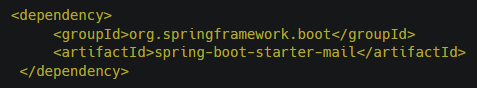
**6. Spring Boot Starter WebSocket**

Purpose: Supports WebSocket communication for real-time interaction between server and client.



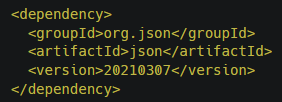
**7. Spring Boot Starter Mail**

Purpose: Provides support for sending emails using JavaMailSender.



**8. JSON Library**

Purpose: Handles JSON data for parsing and generating JSON objects.



# Frontend Visualization of the Water Quality Monitoring System

The frontend of the Water Quality Monitoring System is built using Angular, providing an intuitive and interactive interface for real-time monitoring, historical data analysis, and anomaly detection. Below is a detailed description of each main component and view based on the visual layout of the system.

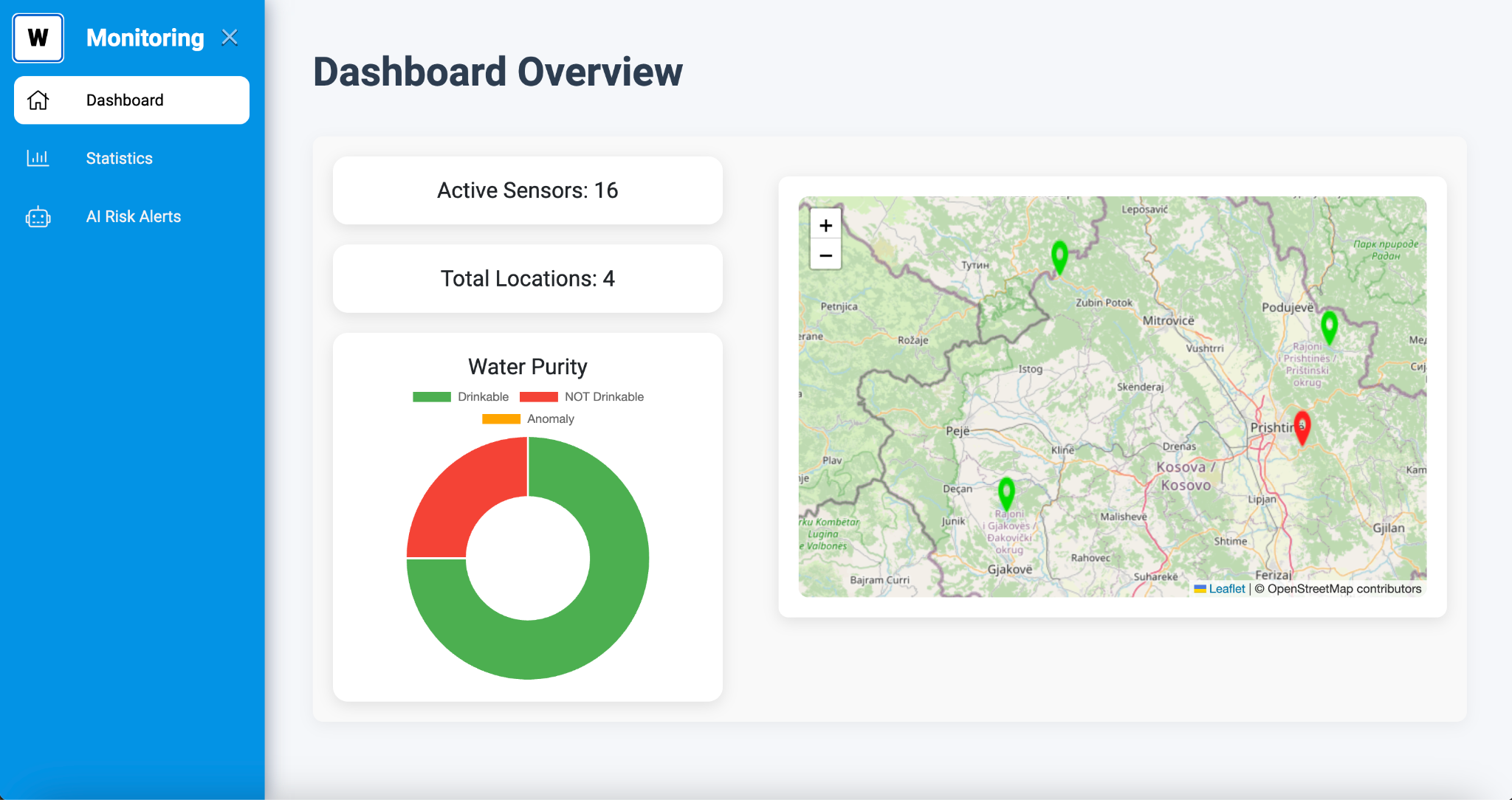
#### **A. Sidebar Navigation**

The application features a sidebar for easy navigation between different views. It includes the following main sections:

* Dashboard: Provides an overview of the water quality status across all monitored locations.
* Statistics: Displays detailed statistical analysis of water quality over time for each location.
* AI Risk Alerts: Shows alerts for any detected anomalies and potential issues.

#### **B. Main Views**

1. Dashboard Overview
   * Description: The Dashboard provides a comprehensive overview of the current status of the water quality monitoring system.
   * Components:
     + Active Sensors: Displays the total number of active sensors currently monitoring water quality parameters.
     + Total Locations: Indicates the number of locations being monitored, which in this case is 4.
     + Water Purity Pie Chart: Visualizes the distribution of water quality classifications—Drinkable, Not Drinkable, and Anomaly. The chart provides an at-a-glance view of the overall water quality status across all locations.
     + Interactive Map: Displays a map with markers representing the monitored locations. Each marker is color-coded to indicate the water quality status:
       - Green: Drinkable water.
       - Red: Not Drinkable water.
       - Yellow: Anomaly detected.
     + Functionality: Users can click on a location marker to view detailed information about the latest readings for that specific location, including temperature, pH, and turbidity levels.



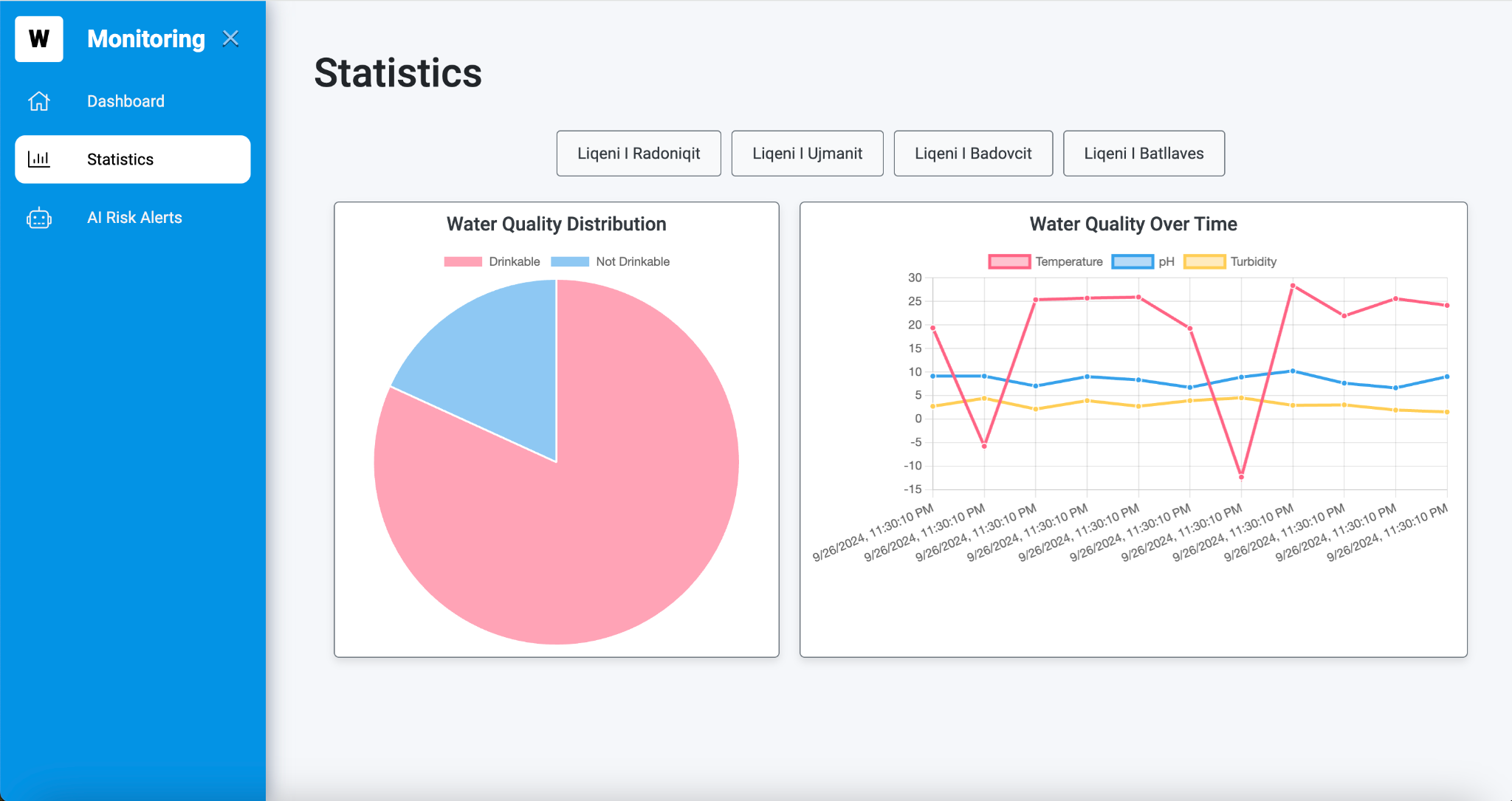
*Figure 7. Dashboard*

1. Location Details on Map
   * Description: When a user clicks on a location marker, a detailed tooltip appears, displaying the following:
     + Location Coordinates: Shows the latitude and longitude of the selected location.
     + Water Quality Status: Indicates whether the water is Drinkable, Not Drinkable, or if an Anomaly is present.
     + Parameter Readings: Shows the latest readings for key parameters such as:
       - Temperature (°C)
       - pH Level
       - Turbidity
   * Purpose: This feature allows users to quickly assess the water quality at a specific location and take action if necessary.

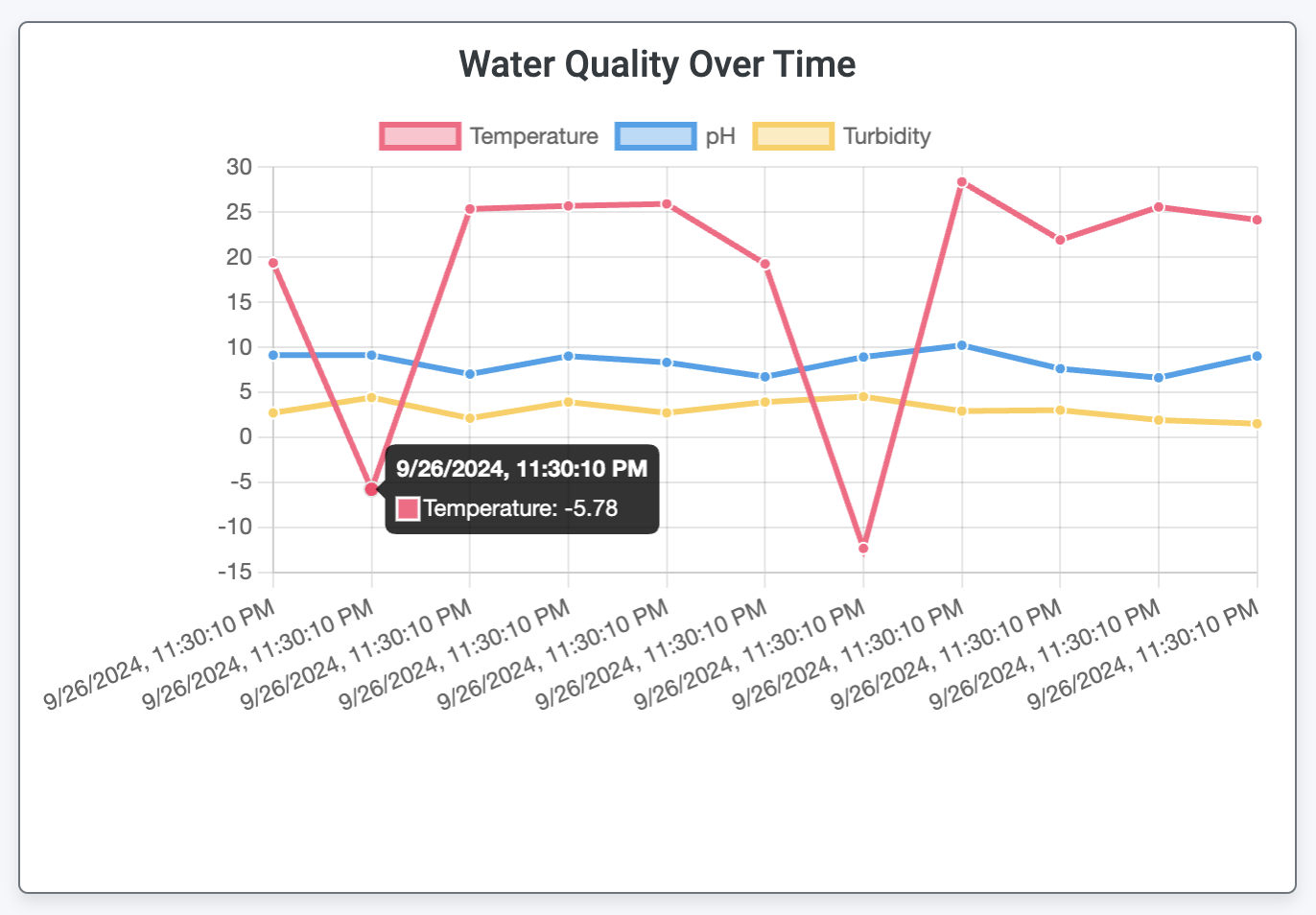


*Figure 8 Locations in Map*

1. Statistics View
   * Description: The Statistics view provides a more in-depth analysis of water quality data, focusing on historical trends and distribution.
   * Components:
     + Location Filters: Four buttons at the top allow users to filter data based on the location: Liqeni i Radoniqit, Liqeni i Ujmanit, Liqeni i Badovcit, and Liqeni i Batllaves.
     + Water Quality Distribution Pie Chart: Shows the proportion of time the water was classified as Drinkable or Not Drinkable at a selected location.
     + Water Quality Over Time Graph: Displays the trends of key parameters (Temperature, pH, Turbidity) over a specified time range. This graph helps in identifying patterns and anomalies over time.
   * Functionality: Users can select different locations to view their specific data, and the graphs update accordingly.



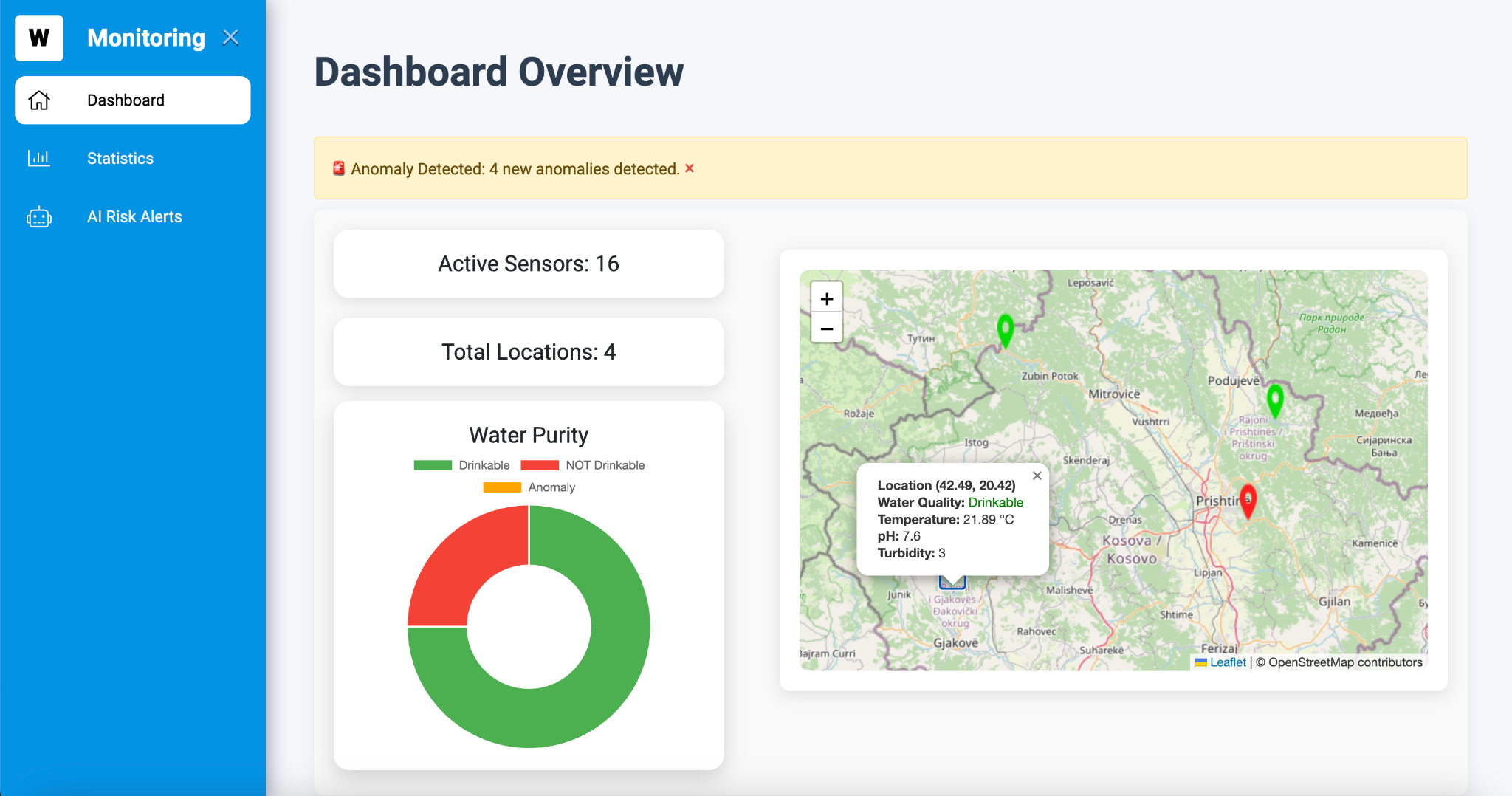
*Figure 9 Statistics based on location*



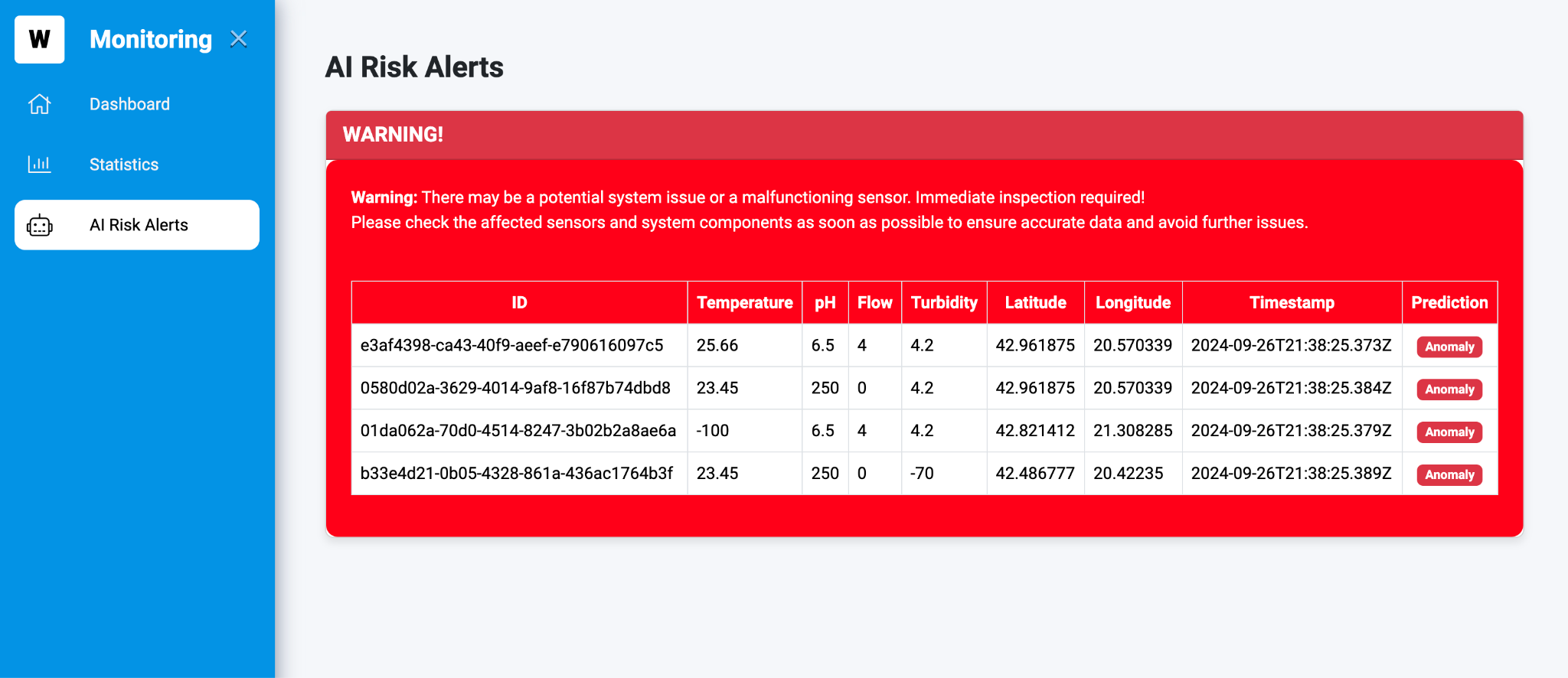
*Figure 10 Parameters data over time*

**C. AI Risk Alerts View**

* + Description: The AI Risk Alerts view highlights potential issues and anomalies detected in the system, requiring immediate attention.
  + Components:
    - Alert Banner: A prominent banner at the top alerts users to the number of new anomalies detected.
    - Anomaly Table: Lists all detected anomalies with detailed information, including:
      * ID: Unique identifier for each anomaly.
      * Temperature, pH, Flow, Turbidity: The parameter readings that triggered the anomaly.
      * Latitude and Longitude: The location of the anomaly.
      * Timestamp: The date and time when the anomaly was detected.
      * Prediction: Indicates the status as an Anomaly.
  + Purpose: This view allows users to quickly identify and investigate potential issues, ensuring timely intervention and maintenance.



*Figure 11 Dashboard Alert*



*Figure 12 AI Risk Alerts*

## 6.1 Technical Implementation

* **Angular**: The frontend is developed using Angular, leveraging its powerful component-based architecture for building a dynamic and responsive user interface.
* **Leaflet**: The map component uses Leaflet, a leading JavaScript library for interactive maps, to display location markers and provide location-based insights.
* **Chart.js:** This library is used for creating the pie charts and line graphs, providing an interactive and visually appealing way to present data.
* **REST API Integration:** The frontend communicates with the backend services through RESTful APIs, enabling real-time data retrieval and updates.

# Artificial Intelligence (AI) Implementation

The AI implementation within the water quality monitoring system is designed to provide predictive analysis and anomaly detection, ensuring the safety and drinkability of water based on real-time sensor data. The AI model used is a Logistic Regression classifier, trained on a historical dataset consisting of 100,000 sensor data points, including attributes like temperature, pH, flow, and turbidity.

## 7.1 Data Collection and Preparation

The foundational step in the AI implementation is the collection and preparation of a comprehensive dataset. This dataset contains 100,000 records, each representing a snapshot of water quality at a specific point in time, across various sensor locations.

* **Attributes in the Dataset:**
  + **Temperature (°C):** The temperature of the water, which can affect biological and chemical processes.
  + **pH Level:** A measure of the acidity or alkalinity of the water, critical for maintaining a balanced aquatic environment.
  + **Flow Rate (L/min):** The volume of water passing through a sensor location, indicating the movement and distribution of water.
  + **Turbidity (NTU):** The clarity of the water, with higher values indicating more suspended particles, potentially signaling contamination.
* **Class Labels:**
  + The dataset is labeled with two classes: "Drinkable" (1) and "Not Drinkable" (0). These labels are crucial for training the Logistic Regression model, allowing it to learn the relationship between sensor readings and water quality.
* **Data Preprocessing:**
  + The class labels are converted from numeric to nominal values using the NumericToNominal filter in Weka. This conversion is necessary for compatibility with the Logistic Regression model, which expects class labels to be in nominal format.

#### 

## 7.3 Model Training

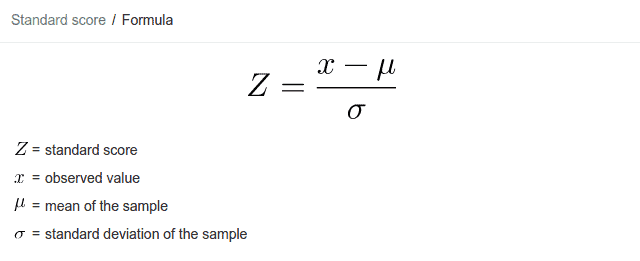
The Logistic Regression model is trained using the historical dataset. Logistic Regression is chosen for its simplicity and effectiveness in binary classification problems, making it ideal for predicting whether water is "Drinkable" or "Not Drinkable" based on the input features.

* **Training Process:**
  + The model is fed with the training data, which includes the four attributes (temperature, pH, flow, turbidity) and the corresponding class labels.
  + During training, the model learns the coefficients (weights) that best separate the two classes based on the input features. It uses the sigmoid function to calculate the probability that a given instance belongs to the "Drinkable" class.
  + The training process minimizes the logistic loss function to find the optimal set of weights that result in the highest classification accuracy on the training data.
* **Model Validation:**
  + After training, the model is validated using a subset of the data not seen during training. This step ensures that the model generalizes well to unseen data and is not overfitting the training set.

## 7.4 Real-time Prediction:

Once the model is trained, it is deployed into the Spark Streaming application, where it classifies each new data instance in real-time.

* **Integration with Spark Streaming:**
  + As new sensor readings arrive in real-time, they are processed and converted into the appropriate format expected by the model.
  + For each data instance, the model calculates the probability of the water being drinkable. If the probability is above a certain threshold (e.g., 0.5), the model predicts "Drinkable"; otherwise, it predicts "Not Drinkable".
* **Anomaly Detection Using Z-score:**
  + In addition to classification, the system employs a Z-score-based anomaly detection mechanism. The Z-score measures the number of standard deviations a data point is from the mean of the dataset.

**Formula for Z-score:**

* + Where:
    - XXX is the observed value.
    - μ\muμ is the mean of the dataset.
    - σ\sigmaσ is the standard deviation of the dataset.
  + **Threshold Setting:**
    - A Z-score threshold of 3 is used, meaning that any data point with a Z-score greater than 3 or less than -3 is considered an anomaly. This threshold is chosen based on statistical principles, where 99.7% of data in a normal distribution lies within 3 standard deviations of the mean. Hence, values outside this range are highly likely to be outliers.

#### 

## 7.5 Self-Learning and Model Adaptation

One of the key features of the system is its ability to adapt and learn from new data continuously, enhancing the model’s performance over time.

* **Real-time Dataset Updating:**
  + After each prediction, the new sensor data, along with the predicted class label, is appended to the original dataset (CSV file).
  + This continuous addition of new data allows the model to be retrained periodically with an updated dataset that reflects the latest environmental conditions and sensor behaviors.
* **Retraining and Improvement:**
  + The system can periodically retrain the model with the expanded dataset, incorporating new patterns and improving its predictive accuracy.
  + By incorporating real-time data, the model becomes more robust and adaptive to changing conditions, reducing the likelihood of false predictions and improving anomaly detection.

#### 

## 7.6 AI Workflow in the Spark Streaming System:

1. **Data Ingestion:** Sensor data is ingested in real-time through Kafka and processed into a structured format by Spark Streaming.
2. **Anomaly Detection:** The system calculates the Z-score for each data instance. If the Z-score exceeds the threshold, the data is flagged as an anomaly.
3. **Prediction:** The Logistic Regression model predicts whether the water is "Drinkable" or "Not Drinkable" based on the input features.
4. **Self-Learning:** The newly predicted data is appended to the dataset for future retraining of the model, ensuring continuous learning and adaptation.
5. **Storage and Alerting:** The processed data, along with predictions and anomaly flags, is stored in the Cassandra database. If an anomaly is detected, the system triggers alerts through the AI Risk Alert menu and sends email notifications.

This detailed AI implementation allows the system to not only classify water quality accurately but also adapt to new conditions, making it a powerful tool for real-time water quality monitoring and predictive analytics.

# Alarm System

The alarm system is a critical component of the water quality monitoring system, providing real-time alerts for any detected anomalies in the sensor data. This system helps ensure timely and appropriate actions to address water quality issues. Below is a detailed explanation of how the alarm process works.

## 8.1 Anomaly Identification

The process begins with the identification of anomalies in the sensor data, which is carried out during the data processing phase in Spark Streaming:

* **Anomaly Detection with Z-score:**
  + Each sensor data point is checked using a Z-score-based method to identify values outside the normal range. A Z-score represents how far a value is from the mean of the data group in terms of standard deviation.
  + If the Z-score for a particular parameter is greater than 3 or less than -3, this value is considered an anomaly. This indicates that the value measured by the sensor is significantly different from the average values, indicating a sudden or unusual change in water quality.

## 8.2 Recording and Storing Anomalies

Once an anomaly is identified, the system records it and stores all relevant details in a dedicated data structure for further reporting and analysis.

* **Anomaly Logging:**
  + Each detected anomaly is logged with complete data, including:
    - The sensor values that triggered the anomalies (e.g., temperature, pH, turbidity, flow).
    - The geographical coordinates of the sensor where the anomaly occurred.
    - The timestamp when the anomaly was recorded.
    - The category of the anomaly (e.g., poor water quality, sensor malfunction, extreme value changes).
* **Database Storage:**
  + The identified anomalies are stored in a dedicated table in the Cassandra database, enabling a complete historical record of anomalies for each sensor and location. This table contains all the data needed for retrospective analysis and trend identification.

## 8.3 Real-time Alarm Generation

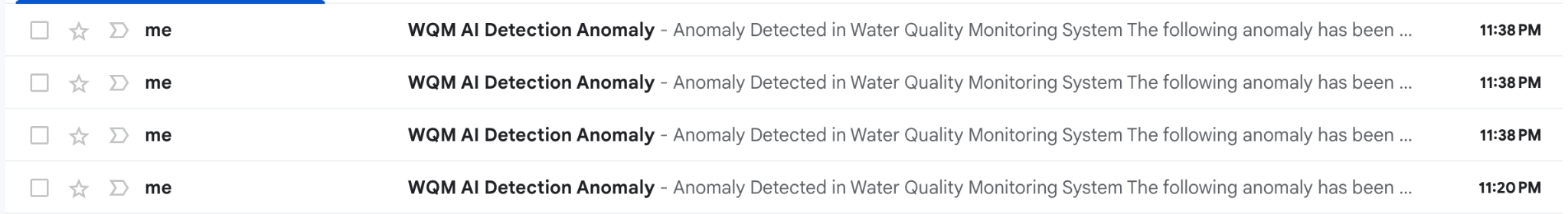
The system generates an alarm the moment an anomaly is detected and communicates it to the user in real-time through various channels:

* **Visual Alarm on the Dashboard:**
  + On the system’s dashboard, in the map of locations, the sensor icon for the corresponding location changes color to yellow to indicate an anomaly (or red for non-drinkable water).
  + An informative pop-up window appears, showing the latest anomaly details for that location.
* **Anomaly Table in the "AI Risk Alert" Menu:**
  + All detected anomalies are automatically added to the "AI Risk Alert" table in the menu. This table contains columns such as anomaly time, location, detected value, and anomaly type.
  + Clicking on a row in the table leads to detailed information about the specific anomaly, including the historical sensor readings before and after the anomaly detection.

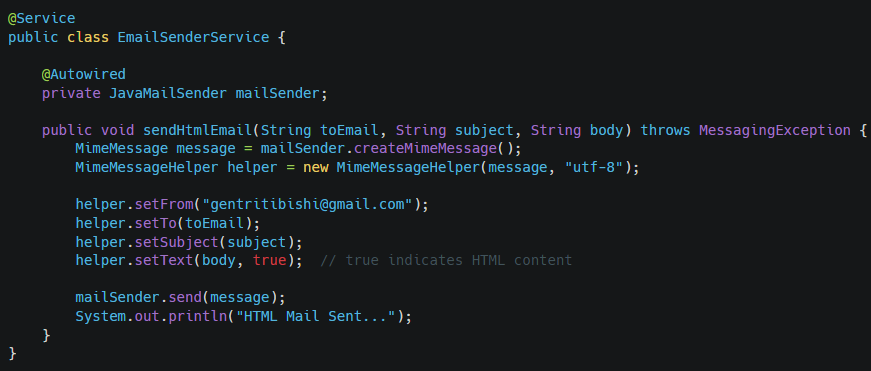
## 8.4 Sending Informative Emails

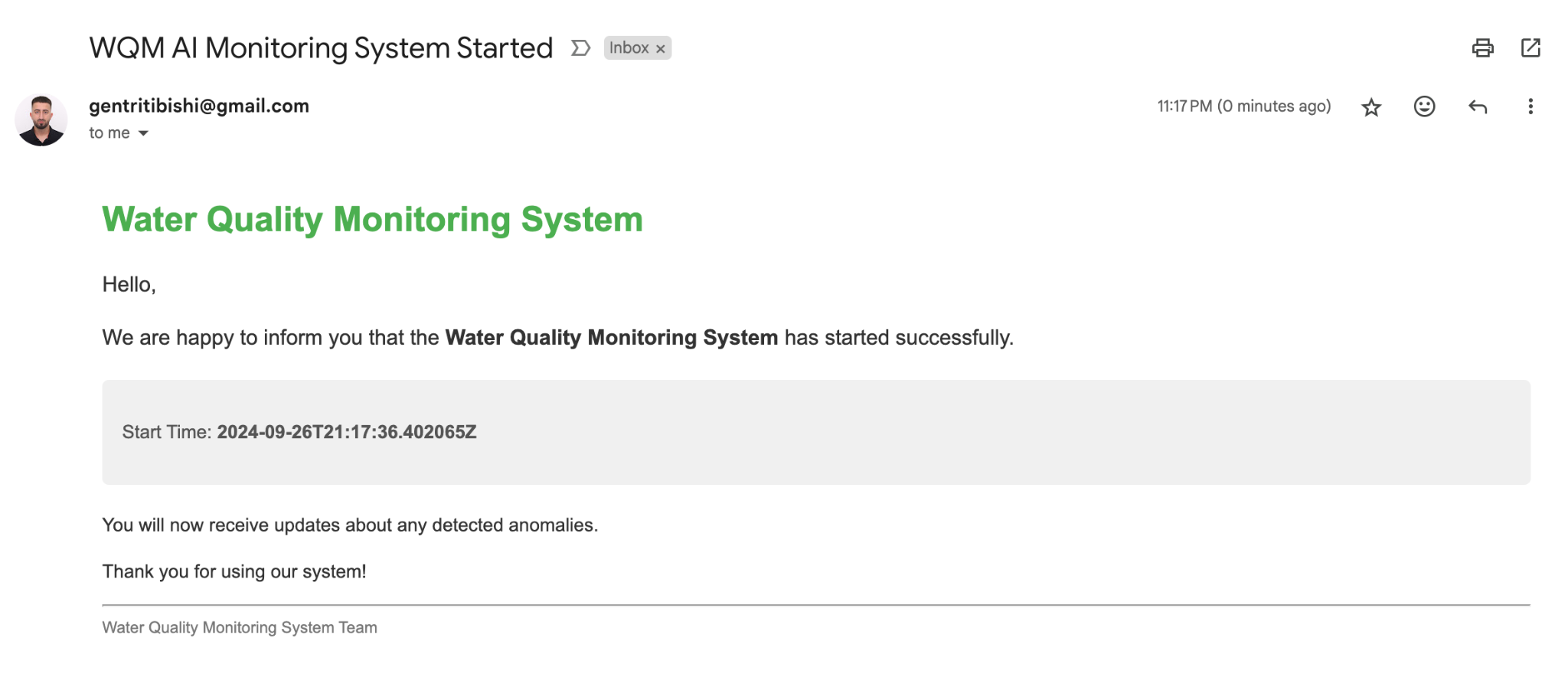
When an anomaly is detected, the system sends an informative email to predefined users, alerting them to the situation.

* **Email Content:**
  + The email contains a summary of the detected anomalies, including:
    - The location of the sensor that detected the anomaly.
    - The exact time of the anomaly detection.
    - The specific values that caused the anomalies.
    - Suggestions for the next steps to be taken (e.g., sensor check, physical sampling for verification).
* **Automatic Sending:**
  + The email is sent automatically at the moment of anomaly detection, ensuring that users are informed in real-time of any sudden changes in water quality.



*Figure 13 Received emails*





*Figure 14 Email for "System Started"*

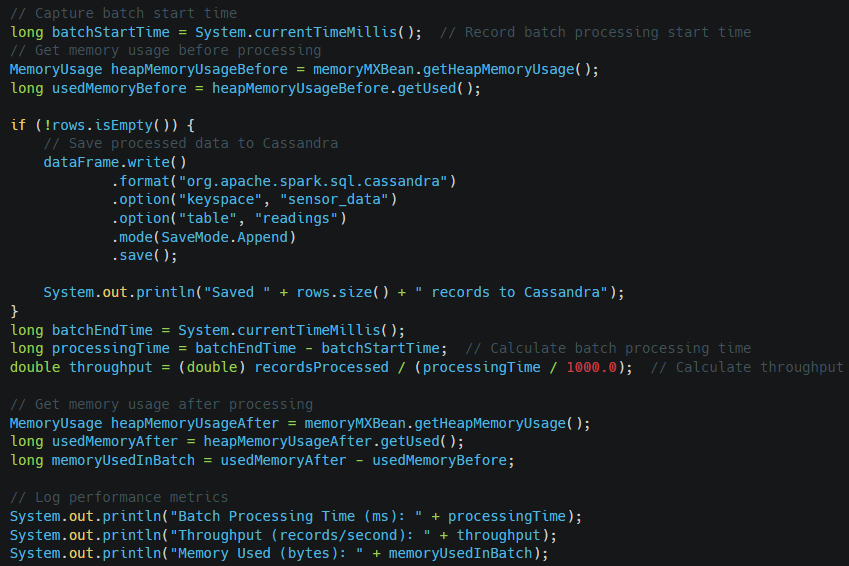


*Figure 15 Email for "Detection Anomaly"*

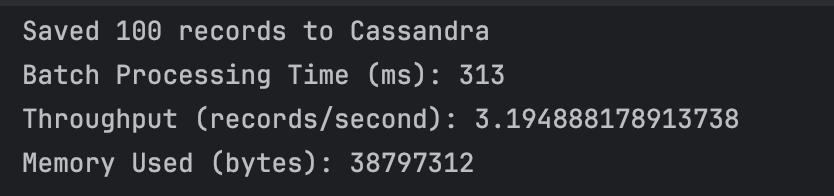
# Performance Analysis and Optimization

The performance analysis in this project focuses on monitoring key performance metrics such as batch processing time, throughput, and memory usage during the data processing stages. This ensures that the system operates efficiently and can handle the incoming data load without performance degradation.

Implementation: In the provided code, performance analysis is carried out by capturing metrics such as the start and end times of batch processing, the number of records processed per second, and memory usage before and after data processing.



1. The batchStartTime captures the time when batch processing begins.
2. MemoryUsage metrics are captured both before and after processing to measure the memory consumed.
3. batchEndTime records the time when batch processing ends.
4. processingTime and throughput are calculated to measure the system's efficiency in handling data.
5. Performance metrics such as batch processing time, throughput, and memory used are logged for monitoring.

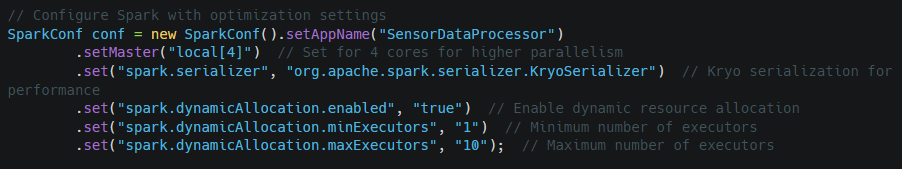


*Figure 16 Performance Analysis*

**B. Optimization**

Optimization in the project focuses on improving performance and resource utilization. Techniques such as adjusting Spark configurations, enabling dynamic resource allocation, and using efficient serialization are employed to ensure the system scales efficiently and processes data with minimal latency.

Implementation: In the code, several configurations are set to optimize the Spark job. These include enabling dynamic resource allocation, setting the number of executors, and using the Kryo serializer for better performance.

****

1. Dynamic Resource Allocation:
   * .set("spark.dynamicAllocation.enabled", "true"): Enables dynamic resource allocation, which allows Spark to scale the number of executors up or down based on workload.
   * .set("spark.dynamicAllocation.minExecutors", "1") and .set("spark.dynamicAllocation.maxExecutors", "10"): These settings define the minimum and maximum number of executors, allowing the system to handle varying loads efficiently.
2. Kryo Serialization:
   * .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer"): Using Kryo serialization helps reduce the serialization overhead and improves performance, especially when handling large datasets.
3. Parallelism:
   * .setMaster("local[4]"): Configures Spark to use 4 cores for higher parallelism, which can speed up the processing of data streams.

# Conclusion

The Water Quality Monitoring System developed in this project effectively integrates advanced technologies, including Apache Kafka for real-time data transmission, Apache Spark Streaming for data processing, and Apache Cassandra for scalable storage, alongside an Angular-based dashboard for user interaction. Throughout the project, we successfully monitored critical water quality parameters such as pH, temperature, turbidity, and flow rates across four significant water bodies in Kosovo. The system’s capabilities for classifying water quality as drinkable, not drinkable, or anomalous, along with its anomaly detection using Z-score methods, exemplify the project’s contribution to enhancing environmental monitoring.

For future improvements, recommendations include enhancing predictive capabilities by exploring advanced AI techniques like deep learning, optimizing performance through automatic scaling solutions based on real-time data load patterns, and expanding the system’s functionalities to incorporate more comprehensive data analyses and visualizations. Additionally, integrating machine learning algorithms for improved anomaly detection accuracy and developing a more robust training dataset that simulates various environmental conditions will further strengthen the system's effectiveness in ensuring water quality safety.