**Project Background and Initial Goals**

At the beginning of our group project, the goal was to create a water quality monitoring system using an Arduino microcontroller and sensors to measure turbidity and temperature. We wanted to build a system that could tell whether water was safe to drink, by analyzing real-time readings from those sensors. The initial idea sounded very practical and exciting, particularly considering the growing global concerns around water safety and accessibility. Our vision was to create an affordable, portable device that could potentially be used in areas with limited access to water testing facilities.

The project aligned with several Sustainable Development Goals, particularly SDG 6 (Clean Water and Sanitation), and we were motivated by the potential real-world applications. We envisioned that, with further development, such a system could be deployed in communities where water quality is a concern, potentially helping prevent waterborne illnesses.

Technical Challenges and Setbacks

However, in reality, this project turned out to be much more technically demanding and frustrating than we expected. The gap between concept and implementation proved to be substantial, particularly given our limited prior experience with sensor technology.

We began by working with a turbidity sensor (SEN0189) and a waterproof temperature sensor (DS18B20). The turbidity sensor was intended to measure the cloudiness or haziness of water, which is a key indicator of contamination, while the temperature sensor would detect whether the water was within a safe drinking range.

Despite spending several hours setting up different wiring configurations, writing and testing various Arduino sketches, and even buying new sensors and components, we continuously ran into major issues. Our initial approach followed standard Arduino tutorials and documentation:

*Initial turbidity sensor setup attempt*

#define TURBIDITY\_SENSOR\_PIN A0

int turbidityValue = 0;

void setup() {

  Serial.begin(9600);

}

void loop() {

  turbidityValue = analogRead(TURBIDITY\_SENSOR\_PIN);

  Serial.print("Turbidity value: ");

  Serial.println(turbidityValue);

  delay(1000);

}

At one point, we connected the turbidity sensor to an analog input, but the output was only ever 1 or 0, regardless of the water sample used. After consulting additional resources, we switched to digital input and adjusted the code:

*Revised attempt with digital pin*

#define TURBIDITY\_SENSOR\_PIN 7

int turbidityValue = 0;

void setup() {

  Serial.begin(9600);

  pinMode(TURBIDITY\_SENSOR\_PIN, INPUT);

}

void loop() {

  turbidityValue = digitalRead(TURBIDITY\_SENSOR\_PIN);

  Serial.print("Turbidity value: ");

  Serial.println(turbidityValue);

  delay(1000);

}

But the readings still made no sense—sometimes showing no change with visibly different water samples or fluctuating wildly with identical samples. We tried calibrating the sensor using distilled water as a reference point, but inconsistent readings persisted.

On the temperature side, we faced issues where the DS18B20 sensor showed -127°C or didn't respond at all. We followed the recommended setup using the OneWire and DallasTemperature libraries:

#include <OneWire.h>

#include <DallasTemperature.h>

#define ONE\_WIRE\_BUS 2

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

void setup() {

  Serial.begin(9600);

  sensors.begin();

}

void loop() {

  sensors.requestTemperatures();

  float tempC = sensors.getTempCByIndex(0);

  Serial.print("Temperature: ");

  Serial.print(tempC);

  Serial.println("°C");

  delay(1000);

}

One setup caused the sensor to overheat, which was alarming—the plastic coating actually began to melt, creating concerns about both the functionality and safety of our prototype. We checked and double-checked our connections, compared our implementation against multiple online tutorials, and even reached out to classmates for assistance, but continued to face persistent issues.

Later, we realized that the sensors weren't fully compatible with the Arduino UNO R4 WiFi board we were using—a critical detail that wasn't clear from the product specifications. We discovered that the turbidity sensor required a specific analog-to-digital conversion resolution that our board wasn't optimized for, and the temperature sensor had communication timing issues with our specific Arduino model.

We then tried to switch to our previously ordered MKR WiFi 1010, thinking its different architecture might solve the compatibility issues. However, this board also didn't work due to voltage incompatibility—our sensors required 5V operation, while the MKR board operated at 3.3V. We attempted to use level shifters to address this voltage mismatch, but that introduced additional complexities and points of failure.

By the time we figured out these fundamental compatibility issues, **a lot of time and energy had been lost**. We had gone through multiple iterations of troubleshooting, component replacement, and code revisions, all to no avail. The practical lab sessions that should have been focused on refining a working prototype were instead consumed by basic troubleshooting.

**Emotional Impact and Team Dynamics**

These constant technical failures caused our team to lose motivation completely. Our project had essentially two distinct parts: the first was getting the hardware working with the sensors providing real values, and the second was analyzing and transmitting that data. By the end of the first part, with nothing working properly despite weeks of effort, our team had lost all incentive to continue.

Personally, I felt extremely burned out. I had put a lot of time and effort into setting up and troubleshooting the sensors, and having nothing functional to show for it was really discouraging. The technical failures drained me to the point where I genuinely didn't have any desire or strength left to continue working on this project. This level of burnout was something I hadn't experienced before in my academic career, and it was both physically and mentally exhausting.

As the deadline got closer, we left the project untouched for a while, pushing it to the very end of our timeline. We even **discussed changing the project** to something much simpler that we knew we could accomplish. However, that wasn't allowed according to the project specifications, which added to the pressure we were already feeling. We were caught between impossible technical challenges and strict project requirements, with no clear path forward.

This experience revealed to me how technical setbacks can affect not just progress but also motivation and team dynamics. Despite our frustrations, my teammates showed remarkable resilience and support, which became crucial when we eventually found a way to salvage the project.

**Turning Point and Project Pivot**

At this point, our lecturer Stephen offered us a solution that changed everything. Thanks to his guidance and understanding of our situation, he suggested a crucial pivot that saved our project.

He advised us to stop focusing on the sensor hardware entirely, and instead simulate the data—generating random numbers that represented turbidity and temperature levels—and then concentrate on sending that data to a Google Spreadsheet using an API. This allowed us to keep the core idea of our project (monitoring water quality), without being limited by faulty sensors.

Stephen's intervention was perfectly timed. He recognized that continuing to struggle with incompatible hardware would only lead to more frustration and potentially a failed project. Instead, he suggested we focus on demonstrating the data analysis and connectivity aspects of our system, which were equally important skills to develop.

This pivot was initially disappointing, as we had been committed to creating a "real" water quality monitoring system. However, it also offered a valuable lesson in realistic scope adjustment and practical problem-solving.

Thanks to his encouragement and guidance, we were able to regain some momentum and reframe the project in a way that was achievable under the circumstances. The shift in focus also allowed us to explore new technical areas—specifically API integration and cloud data storage—that weren't part of our original plan but proved to be valuable learning experiences.

**My Technical Contributions**

Due to my extreme burnout from the first phase of the project, my teammates stepped up and did the majority of the work in the second phase. They focused on writing the code to generate the random values and connect to the Google Spreadsheet API, handling all the connection and data transmission aspects of the project.

My own contribution to this second phase was more limited but still important: I focused specifically on analyzing the simulated data. Although I didn't have the energy or desire to tackle the more complex aspects of the project, I was able to implement the logic that would interpret the generated values.

I implemented the following code for analyzing water quality based on turbidity and temperature readings:

// --- DEFINE YOUR THRESHOLDS HERE ---

const long TURBIDITY\_CLEAR\_THRESHOLD = 30; // Example: Turbidity < 30 is "clear"

const float TEMP\_COLD\_THRESHOLD = 15.0; // Example: Temp < 15C is "cold"

const float TEMP\_GOOD\_MAX\_THRESHOLD = 25.0; // Example: Temp <= 25C (and >=15C) is "good"

// Temperatures > TEMP\_GOOD\_MAX\_THRESHOLD will be "warm"

// --- END OF THRESHOLDS ---

// For simulation purposes, we generated random values

long turbidity = random(0, 30); // Range 0 to 100

// clear water 0 to 10,

// 30 to 70

// coffee 70 to 100

float temperature = random(15, 35); // Range 0 to 100 C

// Determine Temperature Status

if (temperature < TEMP\_COLD\_THRESHOLD) {

  temperatureStatus = "cold";

} else if (temperature <= TEMP\_GOOD\_MAX\_THRESHOLD) {

  temperatureStatus = "good";

} else {

  temperatureStatus = "warm";

}

Serial.print("Temperature Status: ");

Serial.println(temperatureStatus);

// --- END OF Status Calculation ---

// --- IMPLEMENTED: Your Analysis Snippet ---

String finalResult; // This will hold SAFE, NOT IDEAL, or UNSAFE

Serial.print("Overall Analysis: ");

if (turbidityStatus == "clear" && temperatureStatus == "good") {

  finalResult = "SAFE";

  Serial.println("Water is SAFE to drink.");

} else if (turbidityStatus == "clear" && (temperatureStatus == "cold" || temperatureStatus == "warm")) {

  finalResult = "NOT IDEAL";

  Serial.println("Water is drinkable but temperature is not ideal.");

} else {

  // Covers cloudy turbidity OR clear but not good/cold/warm

  // (though that last case is unlikely with current logic)

  finalResult = "UNSAFE";

  Serial.println("Water is UNSAFE to drink.");

}

// --- END OF Analysis Snippet ---

My contribution established the thresholds for water quality assessment and created the decision-making logic to determine whether water samples should be classified as safe, not ideal, or unsafe for consumption. While the implementation was simpler than originally envisioned (since we were working with simulated data rather than real sensor inputs), it still demonstrated the core logic that would be required in a functional water quality monitoring system.

**Project Documentation and Demonstration**

We also recorded a video to demonstrate the project working. In order to show the system outputting each result type correctly, we deliberately adjusted the range of the random values for turbidity and temperature to ensure we could showcase different water quality classifications. This made the video look as though we were receiving real-time sensor data that matched expected conditions.

For example, in the code we used for the demonstration, we manipulated the random number ranges to guarantee specific outcomes:

// For demonstrating "clear" water (SAFE scenario)

long turbidity = random(0, 10); // Very low turbidity

float temperature = random(15, 25); // Ideal temperature range

// For demonstrating "not ideal" water

long turbidity = random(0, 10); // Clear water

float temperature = random(26, 35); // Slightly too warm

// For demonstrating "unsafe" water

long turbidity = random(70, 100); // Very turbid (like coffee)

float temperature = random(15, 25); // Even with good temperature, still unsafe

This approach allowed us to demonstrate all three possible outcomes (SAFE, NOT IDEAL, and UNSAFE) during our video presentation without having to wait for random chance to generate each scenario naturally. Although we were only simulating the data, the video was a helpful way to show that our logic and analysis would work in a real system if the sensors had functioned properly.

To make the demonstration more engaging, we also included a live view of the Google Spreadsheet, showing how each reading was recorded with timestamps and safety classifications. This highlighted the potential for long-term data collection and analysis, which could be valuable for tracking water quality trends over time.

**Learning Outcomes and Reflection**

Looking back, I have mixed feelings about the project. On one hand, the technical setbacks and sensor failures were incredibly disheartening. They drained all my energy and motivation, and I genuinely struggled to keep going. The project exposed significant gaps in my knowledge of electronics and hardware troubleshooting, and revealed how complex the integration of physical sensors and digital systems can be. The burnout I experienced was profound and affected my ability to contribute to the second phase of the project as much as I would have liked.

On the other hand, despite these challenges, I still learned valuable lessons from the experience. I learned that not all projects go according to plan, and that it's okay to pivot when necessary. I developed a deeper appreciation for the iterative nature of engineering projects and the importance of having contingency plans when working with hardware. Most importantly, I experienced firsthand how crucial team support becomes when facing seemingly insurmountable technical obstacles.

Some specific insights I gained include:

1. The reality of technical projects: Hardware projects often involve unexpected compatibility issues and failures that no amount of preparation can fully anticipate. This is a normal part of the engineering process, not a reflection of one's abilities.
2. The value of simulation: While we initially saw using simulated data as a compromise, I came to recognize that simulation is a legitimate and often necessary step in development, allowing for testing of analysis logic without being blocked by hardware issues.
3. Conditional logic implementation: Developing the water quality classification system improved my ability to translate real-world criteria into programmatic decision structures, even in a simplified form.
4. The importance of support systems: My team's willingness to take on the bulk of the work during my period of burnout demonstrated the value of collaborative teamwork. Without their efforts, particularly in implementing the API connection to Google Sheets, the project would not have been completed.

Beyond these insights, I also saw how important timely intervention from mentors can be. Stephen's guidance gave us a clear path forward when we felt completely blocked, demonstrating the value of experienced mentorship in technical projects. His suggestion to pivot rather than continue struggling with incompatible hardware likely saved not just our project but also our motivation to continue in this field.

**Future Improvements and Applications**

If I were to continue this project, several improvements come to mind:

1. Sensor Selection: I would research more thoroughly compatible sensor options for our specific microcontroller, possibly selecting digital sensors with more standardized interfaces like