Spring-Fireward

Diego Iván Morales Gallardo

October 30, 2024

Contents

1 Executive Summary

In response to the critical issue of frequent forest fires in the Bosque de La Primavera in Jalisco, I developed the project *Spring-Fireward*. This initiative leverages the Internet of Things (IoT) to create an integrated system for the prevention, detection, and management of forest fires. Utilizing a combination of sensors—including smoke, temperature/humidity, and water sensors—connected via a microcontroller with internet connectivity (ESP8266), the system collects real-time data and transmits it to a Firebase real-time database. A web application provides a user-friendly interface for monitoring and analyzing this data, facilitating prompt responses to potential fire threats.

2 Introduction

2.1 Context and Justification of the Project

The Bosque de La Primavera, located in Jalisco, Mexico, has suffered from over 500 forest fires in the past 12 years. These fires have caused significant ecological damage, affecting biodiversity and contributing to environmental degradation. The recurring nature of these incidents underscores the need for innovative solutions to prevent and manage forest fires effectively.

Recognizing the potential of IoT technologies in environmental monitoring, I embarked on the *Spring-Fireward* project. By integrating sensors and real-time data analysis, this project aims to enhance early detection capabilities and enable swift intervention, thereby mitigating the impact of forest fires.

2.2 General and Specific Objectives

General Objective:

- To develop an IoT-based system that significantly reduces the incidence and impact of forest fires in the Bosque de La Primavera through early detection, prevention, and effective management.

Specific Objectives:

1. Design and implement a sensor network using an ESP8266 microcontroller with internet connectivity.
2. Integrate smoke, temperature/humidity, and water sensors for comprehensive environmental monitoring.
3. Establish a real-time data connection with a Firebase database for data storage and analysis.
4. Develop a web application for real-time visualization and monitoring of sensor data.
5. Test and validate the system under various conditions to ensure reliability and effectiveness.

2.3 Scope of the Project

The project encompasses the complete development cycle of the *Spring-Fireward* system, including:

- Designing the hardware circuitry and assembling the prototype.
- Programming the microcontroller and sensors for synchronized operation.
- Establishing a real-time connection with a Firebase database.
- Developing a web application for data visualization and user interaction.
- Conducting comprehensive testing to validate system performance.

3 Description of the Problem

3.1 Current Situation of the Bosque de La Primavera

The Bosque de La Primavera is a critical ecological zone in Jalisco, serving as a habitat for diverse flora and fauna. Over the past decade, the forest has experienced an alarming number of fires, exceeding 500 incidents. These fires not only devastate the local ecosystem but also pose health risks to nearby communities due to smoke and air pollution.

3.2 Impact of Forest Fires

Forest fires in the Bosque de La Primavera have led to:

- Loss of biodiversity, including endemic species.
- Soil degradation and increased erosion.
- Release of significant amounts of carbon dioxide, contributing to climate change.
- Economic losses related to tourism and forestry resources.
- Health hazards for local populations due to poor air quality.

3.3 Need for a Technological Solution

Traditional methods of fire detection and prevention rely heavily on human surveillance and delayed reporting, which often result in slower response times. There is a pressing need for a technological solution that can:

- Provide real-time monitoring of environmental conditions.
- Enable early detection of fire indicators such as smoke and elevated temperatures.
- Facilitate rapid communication with emergency services.
- Store historical data to analyze patterns and improve future prevention strategies.

4 Proposed Solution: Spring-Fireward

4.1 General Description of the System

Spring-Fireward is an IoT-based system designed to monitor environmental conditions in real-time to detect and prevent forest fires. The system integrates three primary sensors:

- **Smoke Sensor (MQ-2):** Detects the presence of smoke particles in the air.
- **Temperature and Humidity Sensor (DHT11/DHT22):** Monitors ambient temperature and humidity levels.
- **Water Sensor:** Identifies the presence of water, indicating firefighting efforts or rainfall.

These sensors are connected to an ESP8266 microcontroller, which transmits data to a Firebase real-time database. A web application provides a user interface for monitoring the data and receiving alerts.

4.2 Objectives of Spring-Fireward

- **Early Detection:** Identify potential fire conditions before ignition or in the earliest stages.
- **Real-Time Monitoring:** Continuously track environmental parameters critical to fire risk.
- **Data Analysis:** Store and analyze historical data to identify patterns and improve prevention strategies.
- **User-Friendly Interface:** Provide accessible and actionable information to users and authorities.
- **Scalability:** Design a system that can be expanded to cover larger areas or integrated with additional sensors.

4.3 Expected Benefits

- **Reduced Fire Incidence:** Decrease the number of fires through early detection and intervention.
- **Environmental Preservation:** Protect the biodiversity and ecological balance of the forest.
- **Community Safety:** Enhance the safety of nearby communities by reducing fire-related hazards.
- **Data-Driven Decisions:** Enable authorities to make informed decisions based on real-time and historical data.
- **Resource Optimization:** Improve the allocation of firefighting resources and efforts.

5 Theoretical and Technological Framework

5.1 Internet of Things (IoT) and its Application in Fire Prevention

The Internet of Things refers to the network of physical devices embedded with sensors, software, and connectivity to exchange data. In the context of fire prevention:

- **Real-Time Monitoring:** IoT devices can continuously monitor environmental conditions.

- **Early Warning Systems:** Automated alerts can be generated when certain thresholds are exceeded.
- **Data Collection:** Long-term data accumulation aids in understanding and predicting fire patterns.
- **Remote Access:** Authorities can access data and control systems remotely, facilitating rapid response.

5.2 Microcontrollers with Internet Connectivity (ESP8266)

The ESP8266 is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability. Key features include:

- **Connectivity:** Built-in Wi-Fi allows for internet connectivity without additional modules.
- **Programmability:** Compatible with the Arduino IDE, facilitating programming and integration.
- **GPIO Pins:** Multiple General Purpose Input/Output pins enable connection with various sensors.
- **Low Power Consumption:** Suitable for battery-operated or solar-powered applications.

5.3 Sensors Used

5.3.1 Smoke Sensor (MQ-2)

- **Function:** Detects combustible gases and smoke.
- **Operation:** Changes in electrical resistance indicate the presence of smoke.
- **Application:** Triggers alerts when smoke is detected, indicating potential fire ignition.

5.3.2 Temperature and Humidity Sensor (DHT11/DHT22)

- **Function:** Measures ambient temperature and relative humidity.
- **Operation:** Uses a thermistor and capacitive humidity sensor to provide digital readings.
- **Application:** Identifies conditions conducive to fires, such as high temperatures and low humidity.

5.3.3 Water Sensor

- **Function:** Detects the presence of water or moisture.
- **Operation:** Measures changes in electrical resistance when in contact with water.
- **Application:** Indicates firefighting efforts (water presence) or natural rainfall.

5.4 Real-Time Databases (Firebase)

Firebase is a platform developed by Google for creating mobile and web applications. Features include:

- **Real-Time Database:** Allows for real-time data synchronization across clients.

- **Scalability:** Handles large volumes of data and simultaneous connections.
- **Security:** Provides authentication and data validation rules.
- **Integration:** Compatible with various programming languages and platforms.

5.5 Development of Web Applications for Monitoring

A web application serves as the user interface for the system, providing:

- **Data Visualization:** Displays real-time sensor readings and alerts.
- **User Experience (UX):** Intuitive design for ease of use.
- **Responsive Design:** Accessible from various devices (desktops, tablets, smartphones).
- **Interaction:** Allows users to interpret data and make informed decisions.

6 System Design

6.1 System Architecture

The system architecture comprises:

- **Sensor Network:** Smoke, temperature/humidity, and water sensors connected to the ESP8266 microcontroller.
- **Microcontroller (ESP8266):** Collects sensor data and handles Wi-Fi connectivity.
- **Firebase Real-Time Database:** Stores sensor data and provides real-time synchronization.
- **Web Application:** Interfaces with Firebase to display data and alerts to users.
- **User Devices:** Computers or mobile devices used to access the web application.

6.2 Application Diagram

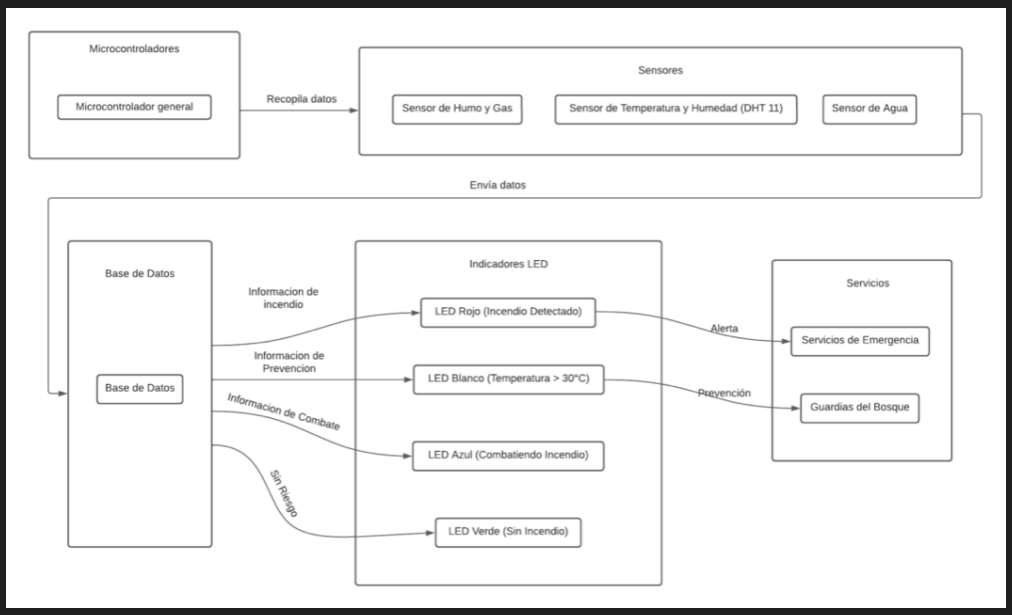

Figure 1: Data flow from sensors to microcontroller, Firebase, and web application.

6.3 Design and Creation of the Database

6.3.1 Entity-Relationship Diagram

I began by creating an Entity-Relationship (ER) diagram to define the structure of the database. The primary entities include:

- **Sensors:** Stores information about each sensor, such as ID, type, and location.
- **Readings:** Records sensor data, timestamps, and status indicators.
- **Alerts:** Logs generated alerts based on specific conditions or thresholds.

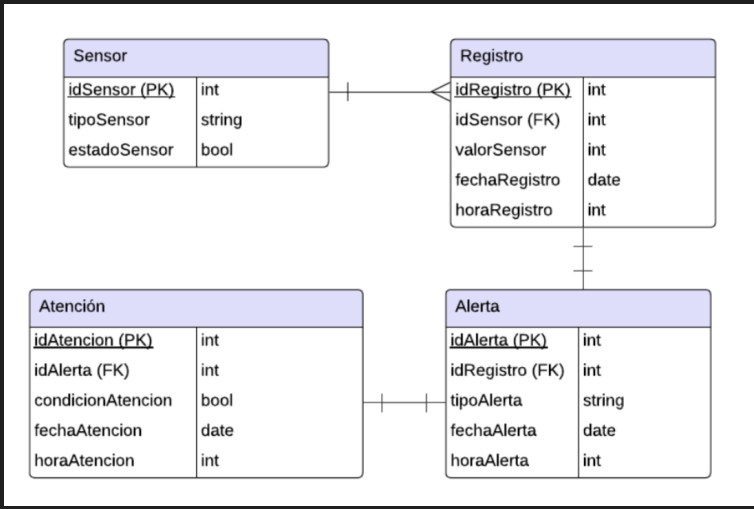

Figure 2: Entity-Relationship Diagram of the Database.

6.3.2 Structure of Tables in Firebase

In Firebase, data is stored in a JSON-like structure. The hierarchy includes:

- `/sensors/`
  - `sensor_id`: Details about the sensor.
- `/readings/`
  - `reading_id`: Contains sensor readings with timestamps.
- `/alerts/`
  - `alert_id`: Information about triggered alerts and their status.

6.4 Web Interface Design

The web application was designed with the following considerations:

- **User Experience (UX):** Simple and intuitive layout for easy navigation.
- **Real-Time Data Display:** Automatic updates of sensor readings without page reloads.
- **Alert Indicators:** Visual cues (e.g., color-coded icons) to represent different alert levels.
- **Responsive Design:** Compatibility with various screen sizes and devices.
- **Accessibility:** Ensuring that the interface is usable by people with varying abilities.

6.5 Circuitry and Physical Prototype

6.5.1 List of Components

- **ESP8266 NodeMCU Microcontroller**
- **MQ-2 Smoke Sensor**
- **DHT11 Temperature and Humidity Sensor**
- **Water Detection Sensor**
- **Breadboard and Jumper Wires**
- **LED Indicators (Red, Yellow, Green, Blue)**
- **Protective Enclosure (Acrylic Box)**
- **Resistors and Capacitors as needed**

6.5.2 Circuit Schematic

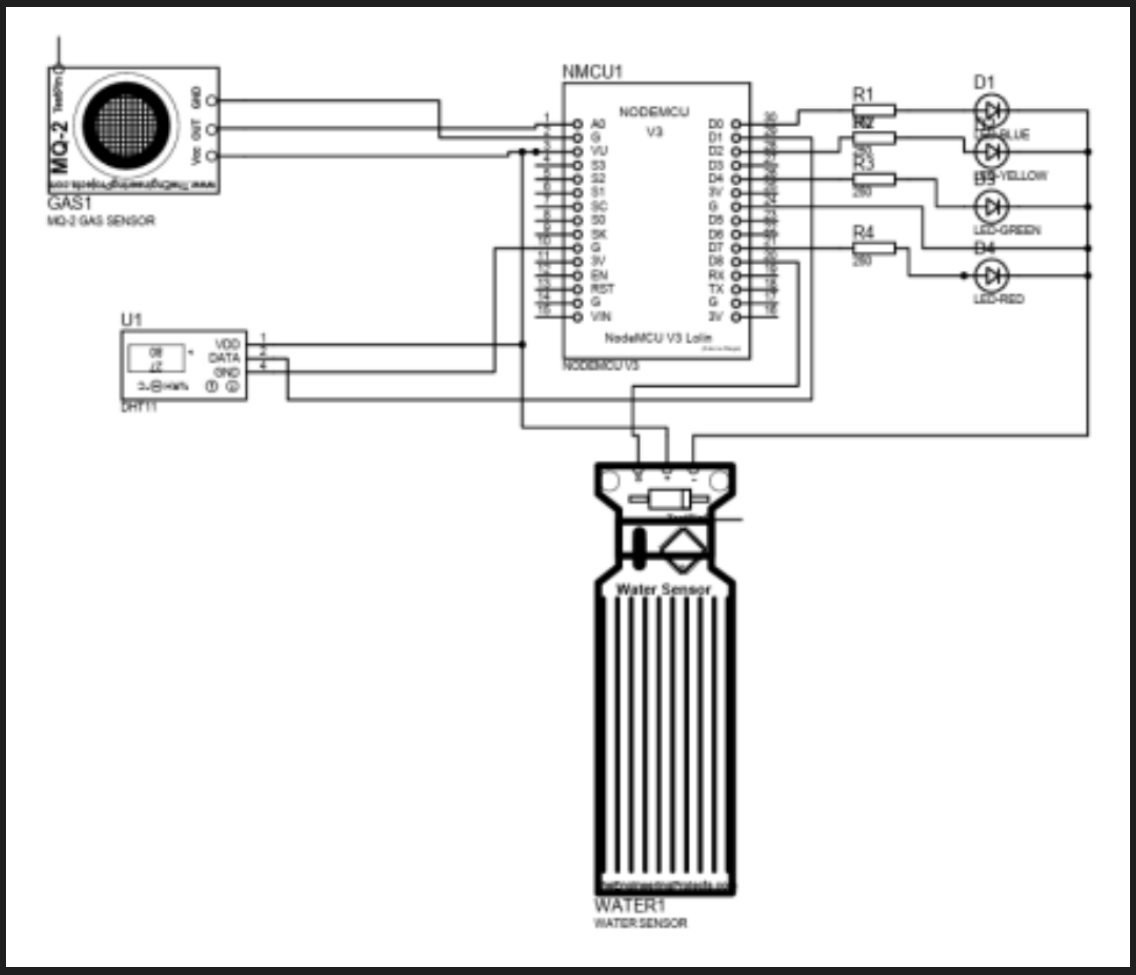

Figure 3: Circuit schematic showing connections between the microcontroller and sensors.

By integrating these components and following this design, I developed a functional prototype capable of monitoring environmental conditions and providing timely alerts to aid in forest fire prevention and management.

7 Implementation

7.1 Configuration and Initialization of Firebase

To establish real-time data synchronization and storage, I configured and initialized Firebase Realtime Database for the project. I began by including the necessary libraries and defining the Firebase credentials in the Arduino code:

```
// Library configuration for ESP32 and ESP8266
#if defined(ESP32)
  #include <WiFi.h>
  #include <FirebaseESP32.h>
#elif defined(ESP8266)
  #include <ESP8266WiFi.h>
  #include <FirebaseESP8266.h>
  #include <WifiUDP.h>
  #include <NTPClient.h>
  #include <Time.h>
  #include <TimeLib.h>
  #include <Timezone.h>
#endif

// Firebase library addons
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>

// WiFi credentials
const char* WIFI_SSID = "Diego";
const char* WIFI_PASSWORD = "12345678";

// API information and Firebase credentials
#define API_KEY "AIzaSyDK1j9W3CmFYRXXfmwoxa0gWj2ykM4Ztys"
#define DATABASE_URL "https://spring-fireward-a153a-default-rtdb.firebaseio.com"
#define USER_EMAIL "a01643382@tec.mx"
#define USER_PASSWORD "123456"

// Firebase objects
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
```

Listing 1: Arduino Code with ESP32 and ESP8266 Firebase Configuration

I initialized Firebase in the `setup()` function by configuring the API key, user credentials, and database URL:

```
void setup() {
    // Initialize Serial Monitor
    Serial.begin(9600);

    // Connect to Wi-Fi
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("Connected to WiFi. IP address: ");
    Serial.println(WiFi.localIP());

    // Configure Firebase
    config.api_key = API_KEY;
    auth.user.email = USER_EMAIL;
    auth.user.password = USER_PASSWORD;
    config.database_url = DATABASE_URL;
    config.token_status_callback = tokenStatusCallback; // Optional
    Firebase.begin(&config, &auth);
}
```

Listing 2: Arduino Setup Function for Wi-Fi and Firebase Configuration

This setup allowed the ESP8266 microcontroller to authenticate and communicate with the Firebase database securely over Wi-Fi.

7.2 Programming the NodeMCU (ESP8266)

7.2.1 Connection to Wi-Fi and Firebase

I programmed the ESP8266 to connect to a Wi-Fi network and authenticate with Firebase using the credentials defined earlier. The connection process is handled in the `setup()` function, as shown above.

7.2.2 Reading Sensors

In the `loop()` function, I included code to read data from the smoke sensor (MQ-2), temperature/humidity sensor (DHT11), and water sensor. The sensors were initialized and configured appropriately:

```
// Pin definitions and sensor configuration
#define PIN_SENSOR_DHT 05
#include <DHT.h>
#define DHTTYPE DHT11 // DHT 11
DHT dht(PIN_SENSOR_DHT, DHTTYPE);

// Sensor variables
int water;
int LED = 16;          // Water LED
int LED_YELLOW = 04;   // Temperature LED
```

```
int LED_RED = 13;      // Fire LED
int LED_GREEN = 02;    // All good LED
int smoke = A0;
int sensorThres = 100;

void setup() {
    // Sensor initialization
    dht.begin();
    pinMode(LED_YELLOW, OUTPUT);
    pinMode(LED, OUTPUT);
    pinMode(smoke, INPUT);
    pinMode(LED_RED, OUTPUT);
    pinMode(LED_GREEN, OUTPUT);
    // Other setup code...
}
```

Listing 3: Arduino Code for Pin Definitions and Sensor Configuration

In the `loop()` function, sensor readings are obtained:

```
void loop() {
    // Read temperature and humidity
    float temperature = dht.readTemperature();
    float humidity = dht.readHumidity();

    // Read smoke level
    int analogSensor = analogRead(smoke);

    // Read water detection
    water = analogRead(15);

    // Debugging outputs
    Serial.print("Temperature: ");
    Serial.println(temperature);
    Serial.print("Humidity: ");
    Serial.println(humidity);
    Serial.print("Smoke Level: ");
    Serial.println(analogSensor);
    Serial.print("Water Level: ");
    Serial.println(water);

    // Send data to Firebase
    // ...

    // Delay before next reading
    delay(20000);
}
```

Listing 4: Arduino Loop Function for Sensor Readings and Debugging Outputs

7.2.3 Sending Data to Firebase

After reading the sensor values, I formatted the data and sent it to Firebase:

```
 // Send data to Firebase
if (Firebase.RTDB.setFloat(&fbdo, "/readings/temperature", temperature)) {
    Serial.println("Temperature updated successfully");
} else {
    Serial.println("Failed to update temperature");
    Serial.println(fbdo.errorReason());
}

if (Firebase.RTDB.setFloat(&fbdo, "/readings/humidity", humidity)) {
    Serial.println("Humidity updated successfully");
} else {
    Serial.println("Failed to update humidity");
    Serial.println(fbdo.errorReason());
}

if (Firebase.RTDB.setInt(&fbdo, "/readings/smokeLevel", analogSensor)) {
    Serial.println("Smoke level updated successfully");
} else {
    Serial.println("Failed to update smoke level");
    Serial.println(fbdo.errorReason());
}

if (Firebase.RTDB.setInt(&fbdo, "/readings/waterLevel", water)) {
    Serial.println("Water level updated successfully");
} else {
    Serial.println("Failed to update water level");
    Serial.println(fbdo.errorReason());
}
```

Listing 5: Arduino Code for Sending Data to Firebase

This code continuously updates the database with the latest sensor readings.

7.2.4 Handling Alerts

The system checks for certain conditions to trigger alerts:

```
 // Threshold values
int temperatureThreshold = 31; // Degrees Celsius
int smokeThreshold = 430;      // Analog value

// Check for high temperature
if (temperature >= temperatureThreshold) {
    digitalWrite(LED_YELLOW, HIGH);
    Firebase.RTDB.setString(&fbdo, "/alerts/temperature", "High Temperature");
} else {
    digitalWrite(LED_YELLOW, LOW);
```

```
}

// Check for smoke detection
if (analogSensor > smokeThreshold) {
    digitalWrite(LED_RED, HIGH);
    Firebase.RTDB.setString(&fbdo, "/alerts/smoke", "Smoke Detected");
} else {
    digitalWrite(LED_RED, LOW);
}

// Check for water detection after a fire alert
if (digitalRead(LED_RED) == HIGH && water >= 1023) {
    digitalWrite(LED, HIGH); // Water LED
    digitalWrite(LED_RED, LOW);
    Firebase.RTDB.setString(&fbdo, "/alerts/water", "Water Detected");
} else {
    digitalWrite(LED, LOW);
}

// Update green LED status
if (analogSensor >= smokeThreshold || digitalRead(LED_YELLOW) == HIGH || digitalRead(LED_RED)
     == HIGH || digitalRead(LED) == HIGH) {
    digitalWrite(LED_GREEN, LOW);
} else {
    digitalWrite(LED_GREEN, HIGH);
}
```

Listing 6: Arduino Code for Threshold Checks and Alerts

7.3 Development of the Web Application

7.3.1 Visualization of Indicators

I developed a simple web server on the ESP8266 to serve an HTML page displaying the sensor data and system status. The server is set up in the `setup()` function:

```
#include <ESP8266WebServer.h>
ESP8266WebServer server(80);

void setup() {
    // ... Previous setup code

    // Start the web server
    server.on("/", HTTP_GET, handleRoot);
    server.begin();
    Serial.println("Web server started");
}
```

Listing 7: Arduino Code for Setting Up Web Server on ESP8266

The `handleRoot()` function generates the HTML content:

```
void handleRoot() {
    String html = "<html><head>";
    html += "<meta http-equiv='refresh' content='5'>";
    html += "<style>";
    html += "body {text-align: center; background-color: #f0f0f0;}";
    html += "h2 {color: #333;}";
    html += "</style>";
    html += "</head><body>";
    html += "<h2>Spring-Fireward Monitoring System</h2>";

    // Display sensor data
    html += "<p>Temperature: " + String(dht.readTemperature()) + " &deg;C</p>";
    html += "<p>Humidity: " + String(dht.readHumidity()) + " \%</p>";
    html += "<p>Smoke Level: " + String(analogRead(smoke)) + "</p>";
    html += "<p>Water Level: " + String(water) + "</p>";

    // Display status based on conditions
    if (digitalRead(LED_RED) == HIGH) {
        html += "<p style='color:red;'>Fire Alert!</p>";
    } else if (digitalRead(LED_YELLOW) == HIGH) {
        html += "<p style='color:orange;'>High Temperature</p>";
    } else if (digitalRead(LED_GREEN) == HIGH) {
        html += "<p style='color:green;'>All Clear</p>";
    } else if (digitalRead(LED) == HIGH) {
        html += "<p style='color:blue;'>Water Detected</p>";
    }

    html += "</body></html>";
    server.send(200, "text/html", html);
}
```

Listing 8: Arduino Code for Handling Root Web Page

7.3.2 Responsive Design and UX/UI

The web page includes basic CSS styling to enhance readability and user experience. The use of color-coded messages helps users quickly interpret the system status. The page automatically refreshes every 5 seconds to display the latest data.

7.4 Integration of Components and Initial Testing

After assembling the hardware and developing the software components, I integrated the system. I connected the sensors to the ESP8266 according to the circuit schematic and uploaded the firmware. Initial testing involved verifying:

- Successful Wi-Fi and Firebase connection.

- Accurate sensor readings being sent to the database.
- Real-time data updates on the web application.
- Correct triggering of alerts based on predefined thresholds.

The integration was successful, and the system operated as intended during the initial tests.

8 Testing and Validation of the System

8.1 Testing Methodology

I employed a structured testing methodology to validate the system's functionality and reliability:

1. **Unit Testing:** Tested individual components (sensors, microcontroller, web app) separately.
2. **Integration Testing:** Verified the interactions between hardware and software components.
3. **System Testing:** Evaluated the complete system under various scenarios.
4. **Acceptance Testing:** Ensured the system met the defined objectives and user requirements.

8.2 Test Cases

I designed four primary test cases to simulate different environmental conditions:

8.2.1 Case 1: All in Order (No Risk)

- **Conditions:** Temperature below 31℃, no smoke detected.
- **Expected Outcome:** Green indicator on the web app; system reports normal status.

8.2.2 Case 2: Fire Risk (High Temperature)

- **Conditions:** Temperature above 31℃, no smoke detected.
- **Expected Outcome:** Yellow indicator; system warns of potential fire risk.

8.2.3 Case 3: Fire Alert (High Temperature and Smoke Detected)

- **Conditions:** Temperature above 31℃, smoke detected.
- **Expected Outcome:** Red indicator; system issues a fire alert.

8.2.4 Case 4: Fire Attended (Water Detection)

- **Conditions:** Water detected after a fire alert.
- **Expected Outcome:** Blue indicator; system acknowledges firefighting efforts.

8.3 Test Results

The system performed as expected in all test cases:

- **Case 1:** The web application displayed normal status; green LED was lit.
- **Case 2:** The web application displayed high temperature warning; yellow LED was lit.
- **Case 3:** The web application displayed fire alert; red LED was lit; alerts were sent to Firebase.
- **Case 4:** Upon detecting water, the system acknowledged firefighting efforts; blue indicator was activated.

8.4 Adjustments and Improvements Made

Based on the testing, I made several adjustments:

- Calibrated the smoke sensor to reduce false positives.
- Optimized the threshold values for temperature based on environmental data.
- Improved the stability of the Wi-Fi connection by adding reconnection logic.
- Enhanced the web application's performance by optimizing the server code.

These improvements increased the system's accuracy and reliability.

9 Project Management

9.1 Project Charter

9.1.1 Project Name

Spring-Fireward

9.1.2 Justification

Due to the high incidence of forest fires in the Bosque de La Primavera, there is an urgent need for an effective monitoring and early warning system. This project aims to leverage IoT technology to address this environmental challenge.

9.1.3 Objectives

General Objective:

To develop an IoT-based system for early detection, prevention, and management of forest fires in the Bosque de La Primavera.

Specific Objectives:

1. Implement a sensor network with internet connectivity.
2. Develop a web application for real-time data visualization.
3. Establish a Firebase database for data storage and analysis.
4. Test and validate the system under various environmental conditions.

9.1.4 High-Level Requirements

- R1: Design a system using an ESP8266 microcontroller.
- R2: Integrate smoke, temperature/humidity, and water sensors.
- R3: Establish a real-time connection with Firebase.
- R4: Develop a user-friendly web application.

9.1.5 High-Level Risks

- Risk 1: Network connectivity issues may disrupt data transmission.
- Risk 2: Sensors may be damaged during a fire event.
- Risk 3: Environmental conditions may affect sensor accuracy.
- Risk 4: Power supply interruptions could halt system operation.

9.1.6 Constraints and Assumptions

Constraints:

- Limited budget for hardware and development.
- Single developer handling all aspects of the project.

Assumptions:

- Reliable Wi-Fi connectivity is available at the deployment site.
- The protective enclosure adequately shields the hardware from environmental hazards.

9.1.7 General Budget

The estimated budget covers hardware components, development tools, and operational costs:

- Hardware: $150
- Development Tools: Free (using open-source software)
- Miscellaneous Expenses: $50
- **Total Estimated Budget: $200**

9.1.8 Identified Stakeholders

- Myself (Developer)
- Local authorities responsible for forest management
- Community members living near the Bosque de La Primavera
- Environmental organizations interested in fire prevention

9.2 Definition of Scope

9.2.1 General Deliverables

- Functional prototype of the IoT-based fire detection system.
- Web application for real-time data visualization.
- Documentation, including user manuals and technical specifications.
- Test reports and validation results.

9.2.2 Functional and Non-Functional Requirements

Functional Requirements:

- The system must monitor temperature, humidity, smoke, and water presence.
- Data must be transmitted to Firebase in real-time.
- The web application must display data and alerts promptly.

Non-Functional Requirements:

- The system should be reliable and operate continuously.
- The interface should be user-friendly and accessible.
- Data transmission should be secure to prevent unauthorized access.

9.3 Identification and Risk Management

9.3.1 List of Identified Risks

1. **Sensor Failure:** Sensors may malfunction due to harsh environmental conditions.
2. **Power Supply Issues:** Power interruptions could disable the system.
3. **Network Connectivity Loss:** Wi-Fi connectivity may be unstable.
4. **Data Security Threats:** Unauthorized access to the database.
5. **Physical Damage:** Hardware could be damaged by wildlife or vandalism.

9.3.2 Risk Response Plan

- **Mitigation Strategies:**
  - Use high-quality, weather-resistant sensors.
  - Incorporate a backup power supply (e.g., battery or solar panel).
  - Implement automatic reconnection logic for Wi-Fi.
  - Secure the database with authentication and encryption.
  - Encase the hardware in a robust protective enclosure.
- **Contingency Plans:**
  - Regular maintenance checks and sensor calibration.
  - Monitor system uptime and receive alerts for any downtime.
  - Keep backup hardware components readily available.

9.4 Detailed Budget

9.4.1 Human Resources

As the sole developer, my time investment includes:

- Research and Planning: 20 hours
- Hardware Assembly: 15 hours
- Software Development: 40 hours
- Testing and Validation: 25 hours
- Documentation: 10 hours

9.4.2 Materials and Equipment

- ESP8266 NodeMCU: $10
- MQ-2 Smoke Sensor: $5
- DHT11 Sensor: $3
- Water Sensor: $2
- Breadboard and Wires: $10
- LEDs and Resistors: $5
- Protective Enclosure: $15
- Miscellaneous Components: $10
- **Total Hardware Cost: $60**

9.4.3 Operational Expenses

- Internet Service (for testing): $20
- Electricity Consumption: $10

- **Total Operational Expenses: $30**

9.5 Communication Plan

9.5.1 Communication Channels

Since I am the sole developer, internal communication is streamlined. For external communication with stakeholders:

- **Email:** For formal updates and documentation sharing.
- **Phone Calls/Meetings:** For discussing progress and addressing concerns.
- **Web Application Dashboard:** Provides real-time updates to stakeholders.

9.5.2 Frequency and Type of Reports

- **Weekly Updates:** Brief summaries of progress sent via email.
- **Milestone Reports:** Detailed reports at the completion of major phases (e.g., prototype completion, testing results).
- **Final Report:** Comprehensive documentation upon project completion.

10 Agile Development Methodology

10.1 Introduction to Scrum

To manage the development process effectively, I adopted the Scrum framework, an agile methodology that promotes iterative progress, collaboration, and adaptability. Scrum consists of specific roles, events, and artifacts designed to enhance project delivery and quality.

10.2 User Stories

I created user stories to capture the requirements from the perspective of end-users and stakeholders. These stories helped me focus on delivering value incrementally.

- **As a forest ranger**, I want to receive automatic alerts of possible fires so that I can intervene quickly and prevent major damage.
- **As an emergency response coordinator**, I want access to a dashboard to visualize the current conditions of the forest to better understand the environment's state.
- **As a system developer**, I want to be able to update the firmware of the sensors remotely to improve and correct the system without physical displacement.

10.3 Product Backlog (Unrefined and Refined)

I compiled a product backlog containing all the user stories and requirements. The backlog was initially unrefined and then refined by prioritizing the items based on their importance and estimated effort.

Unrefined Backlog:

- Implement sensor data collection.
- Develop real-time alerts.
- Create a user-friendly dashboard.
- Enable remote firmware updates.
- Ensure data security and integrity.

Refined Backlog:

1. **High Priority:**
   - Implement sensor data collection.
   - Develop real-time alerts.
   - Create a user-friendly dashboard.
2. **Medium Priority:**
   - Enable remote firmware updates.
   - Ensure data security and integrity.

10.4 Sprint Backlog

I organized the work into sprints, each focusing on delivering specific features.

Sprint 1:

- Set up hardware and sensors.
- Establish Wi-Fi and Firebase connections.
- Implement basic data collection.

Sprint 2:

- Develop real-time alert system.
- Create the web application dashboard.

- Test data visualization.

Sprint 3:

- Implement remote firmware update functionality.
- Enhance security measures.
- Perform system testing and validation.

10.5 Roles and Responsibilities

As the sole team member, I assumed multiple roles:

- **Product Owner:** Defined the project vision and prioritized the backlog.
- **Scrum Master:** Facilitated the Scrum process and removed impediments.
- **Development Team:** Designed, implemented, and tested the system.

10.6 Project Tracking and Control

I used a simple Kanban board to track tasks and progress. Regular reviews at the end of each sprint allowed me to adjust plans and improve the development process.

11 Conclusions and Recommendations

11.1 Achieved Goals

I successfully developed a functional prototype of an IoT-based fire detection system. The system integrates multiple sensors, communicates with a real-time database, and provides a user-friendly web application for monitoring and alerts. The project met the initial objectives of enhancing early detection capabilities and facilitating swift responses to potential fire threats.

11.2 Challenges Faced

- **Hardware Limitations:** Integrating multiple sensors with the ESP8266 required careful management of GPIO pins and power supply.
- **Connectivity Issues:** Ensuring reliable Wi-Fi connectivity in a forest environment posed challenges.
- **Sensor Calibration:** Achieving accurate readings required extensive calibration and testing under different conditions.

11.3 Potential Impact of the Project

The *Spring-Fireward* system has the potential to significantly reduce the incidence and impact of forest fires in the Bosque de La Primavera. By providing real-time monitoring and early alerts, it enables authorities and communities to respond promptly, protecting the environment and human lives.

11.4 Suggestions for Future Work

- **Scale Deployment:** Expand the system to cover larger areas with multiple sensor nodes.
- **Energy Efficiency:** Incorporate solar panels or energy harvesting methods to enhance sustainability.
- **Advanced Analytics:** Implement machine learning algorithms to predict fire risks based on historical data.
- **Mobile Application:** Develop a mobile app for broader accessibility and notifications.

12 Bibliographic References

References

[1] Firebase Documentation. Available at: https://firebase.google.com/docs

[2] ESP8266 NodeMCU Documentation. Available at: https://esp8266-arduino-spanish.readthedocs.io/es/latest/

[3] DHT11 Temperature and Humidity Sensor Datasheet.

[4] MQ-2 Gas Sensor Datasheet.

13 Appendices

13.1 User Manual for the Web Application

The web application can be accessed via a web browser connected to the same network as the ESP8266. The interface displays real-time sensor data and alerts.

Features:

- Automatic updates every 5 seconds.
- Color-coded status indicators.
- Sensor readings for temperature, humidity, smoke, and water levels.

Instructions:

1. Connect your device to the same Wi-Fi network.
2. Open a web browser and enter the IP address displayed in the Serial Monitor.
3. Monitor the sensor data and respond accordingly to alerts.

13.2 Technical Documentation of the Sensors

13.2.1 DHT11 Temperature and Humidity Sensor

Specifications:

- Temperature Range: 0℃ to 50℃
- Humidity Range: 20% to 90% RH
- Accuracy: ±2℃ for temperature, ±5% RH for humidity

13.2.2 MQ-2 Smoke Sensor

Specifications:

- Detects LPG, i-butane, methane, alcohol, hydrogen, and smoke.
- Analog output voltage based on gas concentration.
- Preheat Time: Over 24 hours for accurate readings.

13.3 Detailed Test Results

Case 1: All in Order

- Temperature: 25℃
- Smoke Level: 150
- Water Level: 0
- Status: Green LED lit, ”All Clear” displayed on web app.

Case 2: Fire Risk

- Temperature: 33℃
- Smoke Level: 160
- Water Level: 0
- Status: Yellow LED lit, ”High Temperature” warning displayed.

Case 3: Fire Alert

- Temperature: 35℃
- Smoke Level: 500
- Water Level: 0
- Status: Red LED lit, ”Fire Alert!” displayed, alerts sent to Firebase.

Case 4: Fire Attended

- Temperature: 30℃
- Smoke Level: 450
- Water Level: 1023
- Status: Blue LED lit, ”Water Detected” displayed.

13.4 Fire Response Plan Documentation

In collaboration with local authorities, I developed a basic fire response plan:

Alert Levels:

- **Green:** Normal conditions; continue monitoring.
- **Yellow:** Elevated temperature; increase vigilance.
- **Red:** Fire detected; dispatch firefighting teams immediately.
- **Blue:** Firefighting efforts underway; monitor water levels.

Procedures:

1. Upon receiving a red alert, notify the nearest fire station.
2. Activate community warning systems.
3. Coordinate with emergency services for evacuation if necessary.