- **Project Title**: IoT-Based Fire Detection System with Real-Time Monitoring and Notification
- **Course**: Internet of things(IOT)
- **Instructor**: Amira Henaien
- **Team Members**: Cherif Mohamed Said

  Messaoud Mehdi

  BenSalah Farouk

- **Date of Submission**: 19/01/2025

---

1. Introduction

- **Project Overview**: The project is the construction of an IoT-based fire detection and alert system for fire detection and alerting system. It deploys a set of sensors for the collection of data such as flame presence or higher than normal heat levels that is transmitted via real-time to the web or mobile application for monitoring and if required, management. Providing this solution guarantees that fire hazards are addressed in advance, ensuring safety and reducing the risk of damage.

- **Problem Statement**: Traditional fire detection systems generally cannot be accessed remotely and do not provide real-time monitoring, making it difficult to avert disaster. This project focuses on developing a system for detecting and replying to hazards caused by fire, ensuring all stakeholders receive immediate alerts and eliminating the need for human intervention.

- **Objectives**:

  -IoT-based sensors to detect flame or abnormal heat levels with high precision and use mobile or web applications to send notifications and alerts in real time.

  -Create a feature of allowing remote monitoring and access to the system from anywhere

  -Provide detailed logs of detection events for post-incident analysis.

- **Intended Users and Benefits**:

-System Administrators: Manage the system components and monitor overall performance to ensure optimal functionality.

-Property Owners: Gain peace of mind through real-time updates and the ability to respond swiftly to potential fire threats.
-Emergency Responders: Benefit from accurate and immediate alerts, allowing for faster intervention and reduced damage.

2. Specifications Document (Cahier des Charges)

- **Actors and Roles**:

Actors	**Roles**
Admin	Manages users, sensors, and overall system settings.
Property Owner	Monitors real-time data and receives alerts.

Mobile applications	Display data,allows control,and notifies users of fire or gas risks

Internal Components and Hardware:

-Fire Sensor: Detects flames or abnormal heat and sends signals to the ESP32.
-Gas Sensor: Detects dangerous levels of combustible or toxic gases.
-ESP32 Microcontroller: Processes sensor data and forwards it to external systems.
-Buzzer: Provides immediate on-site alerts for local warnings.
-Firebase Database: Stores detection logs and thresholds for analysis.
-MQTT Broker: Facilitates communication between IoT devices and applications.

- **Functional Requirements**:

-FR1:Alert users via mobile/web applications when flames, abnormal heat, or dangerous gas levels are detected.

-FR2:Display real-time sensor data on a dashboard.

-FR3:Allow users to set and adjust alert thresholds remotely.

-FR4:Trigger an on-site buzzer when fire or gas detection exceeds predefined thresholds.

-FR5:Store data logs for all fire and gas detection events for post-incident analysis.

-FR6:Ensure reliable communication between devices using the MQTT protocol.

-FR7:Offer user-friendly interfaces on mobile and web applications for monitoring and control.

-FR8:Operate under low power consumption to maximize hardware lifespan.

- **Hardware Requirements**:

  -ESP32 Microcontroller: Processes data from sensors and communicates with external systems via Wi-Fi or Bluetooth.

  -Flame Sensor (e.g., IR Flame Sensor): Detects flames or high heat levels.

  -Gas Sensor (e.g., MQ-2 or MQ-7): Detects combustible or toxic gas concentrations.

  -DHT11/DHT22 Sensor: Measures temperature and humidity levels to support detection.

  -Buzzer: Provides on-site audible alerts during fire or gas detection.

  -Power Supply: Powers all hardware components.

  -MQTT Broker Device: Facilitates message exchange between the IoT system and the application.

  -Wi-Fi Router: Provides network connectivity for the ESP32 to communicate with external systems.

-Mobile Device or Computer: Displays sensor data and alerts to the user via the application.

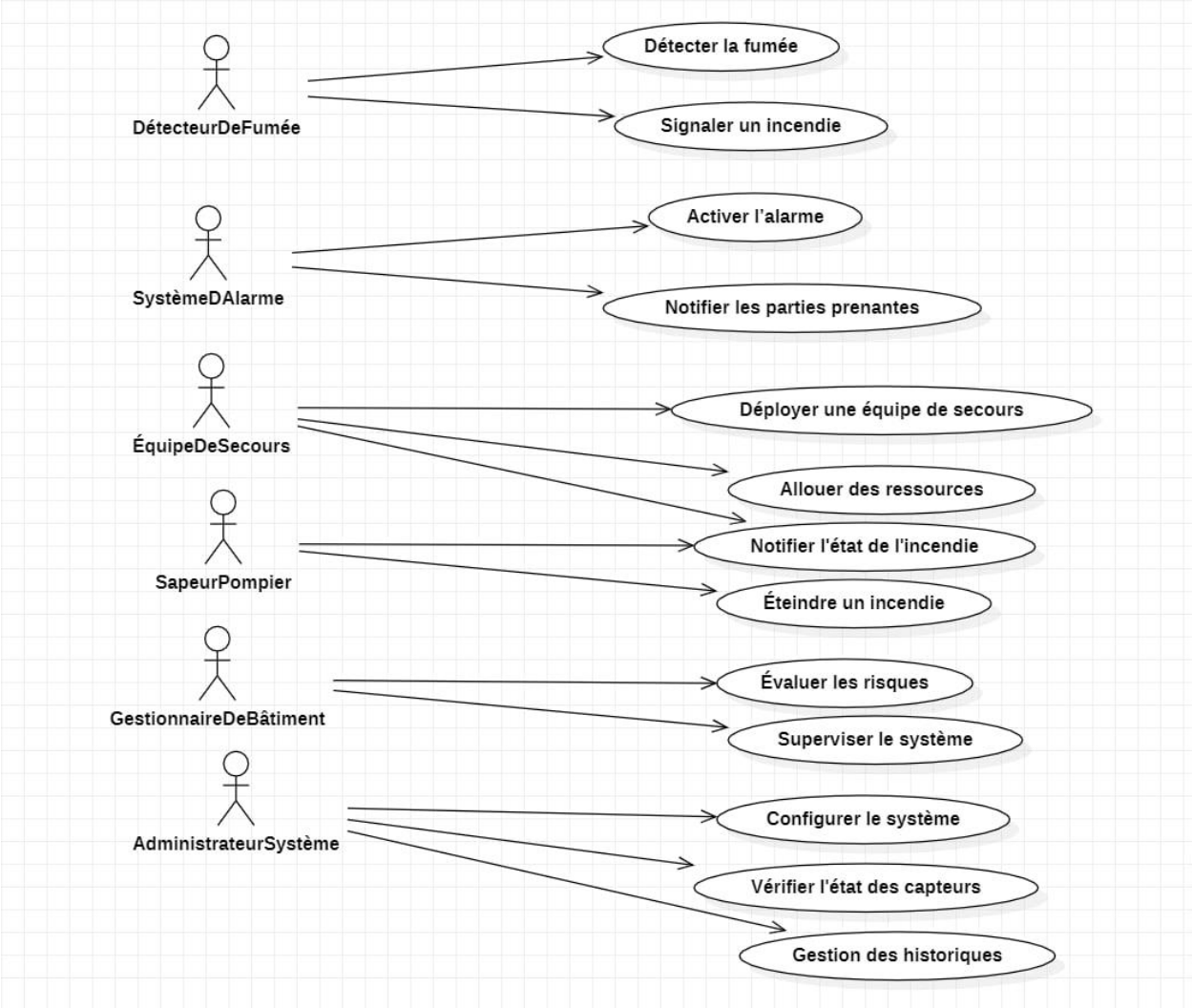
3. System Design and Architecture

- **System Architecture Diagram**: Provide a visual representation of the system architecture, showing the four layers (Perception, Network, Cloud, Application) and data flow between them.
- **Layer Overview**:
  - **Perception Layer**: Handles data collection through sensors and actuators.
  - **Network Layer**: Transmits data from the perception layer to the cloud, ensuring secure communication.
  - **Cloud Layer**: Manages data storage, processing, and access control.
  - **Application Layer**: Provides a user interface for data monitoring, interaction, and alerts.

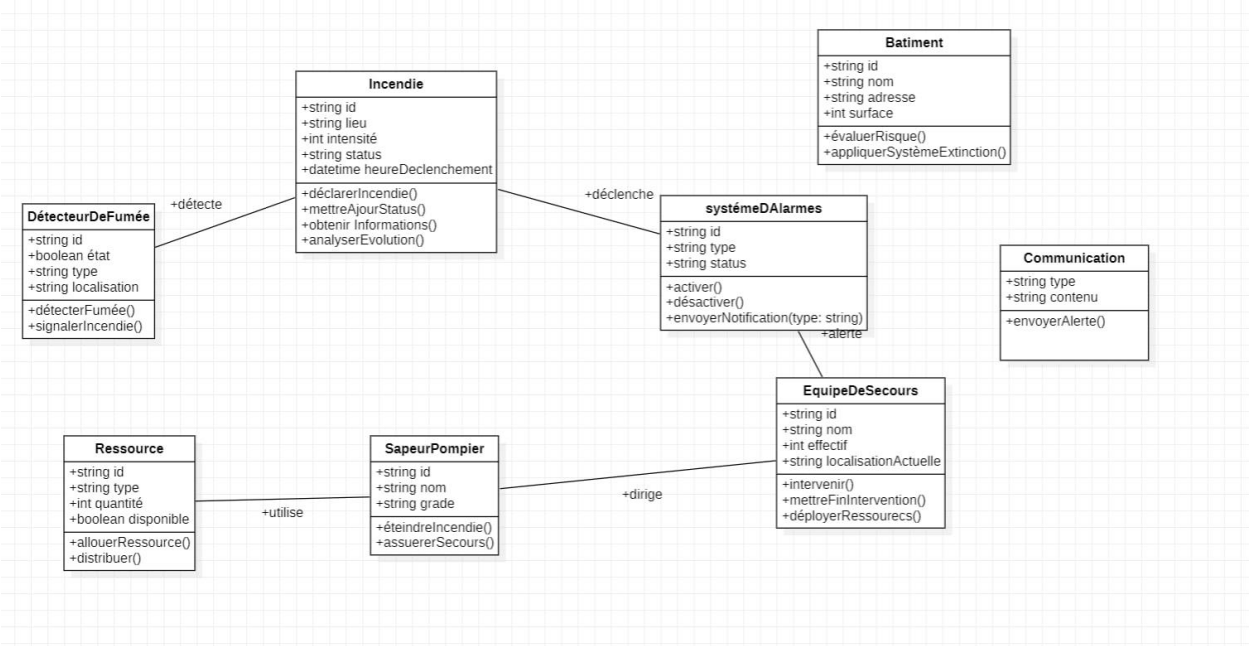
3.1 Conception

Static Conception

- **Use Case Diagram**:



- **Class Diagram**:



4. Implementation by Layers

4.1 Perception Layer

- **Purpose**: This layer collects data through sensors and actuators.
- **Components**:

  -DHT22 Sensor: Measures temperature and humidity.

  -Flame Sensor: Detects flames or abnormal heat levels.
  -Gas Sensor (e.g., MQ-2): Detects dangerous levels of combustible or toxic gases.
  -Buzzer: Provides immediate on-site audible alerts.

- **Data Collection**:
- **Methods**:

  -Sensors are interfaced with the ESP32 microcontroller, which collects raw data.
- **Sampling Intervals**:
  -Data is sampled every 1 second (adjustable based on system requirements).
- **Preprocessing**:
  -Validates sensor data to ensure accuracy.
  -Filters noisy or redundant data to enhance reliability.

4.2 Network Layer

- **Purpose**: Transmits data from the Perception Layer to the Cloud for storage and processing.
- **Communication Protocols**:

  MQTT Protocol:

  -Chosen for its lightweight nature, ideal for IoT messaging.
  -Enables efficient publish-subscribe messaging between devices.
  HTTP:
  -Used for web-based interactions when accessing the cloud or user applications.

- **Connectivity**:

  Wi-Fi:

  -ESP32 connects to a Wi-Fi network for continuous data transmission to the MQTT broker and cloud services.

- **Security Measures**:

  Encryption:

  -Data is encrypted using TLS (Transport Layer Security) during transmission.
  Authentication:
  -Devices authenticate with the MQTT broker using credentials or tokens to prevent unauthorized access.

4.3 Cloud Layer

- **Purpose**: Stores and processes data, enabling remote access.
- **Data Storage**:

  Solution:

  -Firebase Realtime Database is used to store sensor data and logs.
  -Access Methods:
  Data can be accessed via Firebase SDKs or REST API for real-time updates.

- **Data Processing**:

  Transformations:

  -Data is analyzed for threshold violations (e.g., high gas levels or flames detected).
  Analytics:
  -Historical data is aggregated for trend analysis and incident reporting.

- **Access Control**:

  -User authentication and authorization are managed through Firebase Authentication.

  -Users log in using credentials or third-party providers (e.g., Google).
  -Role-based access control (RBAC) ensures that only authorized users can view or manage data.

4.4 Application Layer

- **Purpose**: Provides a user interface for monitoring and controlling the system.
- **User Interface**:

  Dashboard:

  -Displays real-time sensor data (e.g., temperature, gas levels) through visual charts and indicators.
  -Includes an alert system for threshold breaches.
  Mobile App:
  -Allows users to remotely monitor data and adjust thresholds.
  -Sends push notifications for alerts.

5. Privacy and Security Measures

- **Data Security**:

  Transmission Security:

  Encryption:
  -All data transmitted between the Perception Layer, Network Layer, and Cloud Layer is encrypted using TLS (Transport Layer Security) to prevent interception and unauthorized access.
  Secure Protocols:
  - MQTT over TLS is used for IoT device communication.
  - HTTPS is employed for web and mobile app interactions with the cloud.

  Storage Security:
  Data Encryption:
  -Sensor data stored in the Firebase database is encrypted at rest using Firebase's built-in encryption mechanisms.
  Access Logging:
  - Firebase automatically logs access to the database, recording user actions and access attempts for audit and monitoring purposes.

  Access Control:
  Authentication:
  -Firebase Authentication requires users to log in using credentials or trusted third-party providers (e.g., Google or email/password).
  Authorization:

-Role-Based Access Control (RBAC) ensures users have access only to the resources necessary for their role (e.g., admin vs. property owner).

User Privacy:

Data Handling Protocols:

-Sensitive user information, such as login credentials and alert settings, is securely stored and processed in compliance with privacy standards.
-Data retention policies are implemented to automatically delete outdated logs and ensure that data is not stored indefinitely.

Compliance with Privacy Standards:

General Data Protection Regulation (GDPR):
-Ensures the user's right to data access, rectification, and deletion is respected.
User Consent:
-Users must provide explicit consent for data collection, processing, and storage, as per applicable privacy laws.

Minimizing Data Collection:

-The system collects only essential data required for functionality (e.g., sensor readings, user-configured thresholds) to minimize exposure of sensitive information.

User Awareness:

-A clear privacy policy is provided to inform users about how their data is collected, used, and protected.
-Users are notified of any significant changes to the privacy policy or terms of service.

6. Testing and Validation

- **Testing Methods**:

1. Perception Layer Testing

Objective: Verify that sensors accurately capture data and trigger alerts as expected.
Procedures:
  - Simulate fire or gas presence to test flame and gas sensors under various scenarios.
  - Validate DHT22 temperature and humidity readings with calibrated devices.
  - Confirm the buzzer activates when thresholds are exceeded.

Test Cases:
  - Test sensor accuracy by comparing readings with reference devices.

- Verify threshold values trigger appropriate alerts and actions.
- Simulate hardware faults (e.g., disconnect sensors) to ensure the system detects and reports errors.

2. Network Layer Testing
Objective: Ensure reliable and secure data transmission.
Procedures:
- Test MQTT communication for message reliability and latency.
- Simulate network disruptions and observe system recovery.
- Validate TLS encryption for secure data exchange.

Test Cases:
- Measure latency and data loss rates during normal and high-traffic conditions.
- Test reconnection behavior after network interruptions.
- Check that encrypted communication prevents unauthorized access.

3. Cloud Layer Testing
Objective: Validate storage, processing, and access control functionalities.
Procedures:
- Test data storage and retrieval from Firebase.
- Verify data integrity after cloud-based processing.
- Simulate multiple users accessing the system to test authentication and authorization mechanisms.

Test Cases:
- Test data upload and retrieval accuracy.
- Check user roles and permissions for correct access control.
- Measure cloud response times under varying loads.

4. Application Layer Testing
Objective: Test the user interface for usability, functionality, and reliability.
Procedures:
- Test dashboard visualization for real-time updates.
- Simulate user interactions to adjust thresholds and review logs.
- Test notifications on mobile and web platforms.

Test Cases:
- Verify dashboard updates with live sensor data.
- Test alert delivery on different devices and network conditions.
- Ensure threshold adjustments are saved and applied correctly.

5. System Integration Testing
Objective: Ensure all layers interact seamlessly to deliver end-to-end functionality.
Procedures:
- Perform end-to-end testing from sensor data collection to user notification.
- Test simultaneous operations across multiple sensors and devices.

Test Cases:
- Verify data flows correctly from the Perception Layer to the Application Layer.
- Test system behavior under high loads and abnormal conditions.

- **Results and Metrics**: **Accuracy**:

  Flame sensor: 98% detection accuracy in simulated tests.
  Gas sensor: 95% accuracy for detecting hazardous gas concentrations.
  DHT22: ±2% deviation from reference devices.
  Response Time:

Average alert delivery time: 2 seconds.
Dashboard data update frequency: Real-time (<1-second lag).
Uptime:
System uptime during tests: 99.8%.
User Feedback:
Positive feedback on intuitive interface and reliable alert delivery.
Suggestions for additional customization options (e.g., alert tones).

- **Troubleshooting**:

Issues Encountered

High Latency in MQTT Messages: Delays observed in message delivery under high traffic.
  - **Solution**: Optimized MQTT broker configurations and reduced message payload sizes.

Sensor Calibration Errors: Initial readings were inconsistent with reference devices.
  - **Solution**: Recalibrated sensors and adjusted software compensation algorithms.

Data Overwriting in Firebase: Logs were overwritten instead of appended.
  - **Solution**: Modified database write operations to append data with unique timestamps.

Connectivity Issues: ESP32 occasionally failed to reconnect after network disruptions.
  - **Solution**: Added a reconnection algorithm to the ESP32 code for automatic recovery.

7. Challenges and Solutions

- **Challenges Faced**:

1. Perception Layer

Sensor Calibration: Flame and gas sensors showed inconsistent readings during initial tests.
Environmental Noise: Temperature and humidity readings were affected by environmental interference, leading to false positives.
Hardware Durability: Sensors were sensitive to wear and tear during prolonged testing.

2. Network Layer

Connectivity Issues: ESP32 devices sometimes failed to reconnect after Wi-Fi disruptions.
Message Latency: High traffic conditions caused delays in MQTT message delivery.
Security Vulnerabilities: Initial communication lacked sufficient encryption, making data susceptible to interception.

3. Cloud Layer

Data Overwriting: Detection logs in Firebase were overwritten instead of appended.

Scalability: The system faced performance issues when handling large volumes of data.
Access Control: Implementing user-specific permissions proved complex.

4. Application Layer

User Interface Design: Early versions of the dashboard lacked intuitive navigation.
Notification Delays: Alerts on mobile devices experienced intermittent delays.
Customization Requests: Users requested additional customization options for alerts and thresholds.

- **Solutions Implemented**:

1. Perception Layer

Sensor Calibration: Adjusted calibration settings and implemented software algorithms to filter out anomalies.
Environmental Noise Reduction: Introduced additional data validation and smoothing techniques to minimize false positives.
Hardware Upgrades: Replaced low-durability components with more robust alternatives to improve longevity.
2. Network Layer
Improved Connectivity: Added a reconnection algorithm in the ESP32 firmware to ensure automatic recovery from network disruptions.
Optimized MQTT Broker: Reduced message payload sizes and adjusted broker configurations to handle high traffic efficiently.
Enhanced Security: Implemented TLS encryption and authentication protocols for secure data transmission.
3. Cloud Layer
Unique Timestamps: Modified database write operations to include unique timestamps, preventing data overwriting.
Performance Optimization: Migrated to a higher-tier Firebase plan and optimized query performance for scalability.
Access Control Mechanisms: Integrated role-based access control (RBAC) to manage user-specific permissions effectively.
4. Application Layer
Redesigned User Interface: Improved dashboard layout and navigation based on user feedback.
Real-Time Notifications: Optimized notification delivery by implementing push notification services with lower latency.
Customization Features: Added user-defined alert tones, threshold settings, and language preferences to enhance usability.

8. Conclusion and Future Work

- **Project Summary**:

The fire detection IoT project successfully met its objectives by integrating sensors, processing data efficiently, and sending notifications for potential fire hazards. The

system uses ESP32 devices to collect data from DHT22 temperature sensors and transmits the data to a central MQTT broker. The project achieved real-time monitoring and efficient data forwarding to Firebase/ThingSpeak, ensuring that the system can provide alerts when certain fire-related conditions are met. The architecture was implemented using solid IoT and cloud technologies, and the system was tested in various conditions, providing promising results for fire detection in a real-world setting.