# 

# Requirement Specification Document

## 1. Project Overview

Project Name: SAFESNIFF  
Project Type: IoT-based Gas Leakage Detection and Alert System  
Objective: Detect a wide range of gases (including methane, LPG, and smoke) in real-time, and trigger both local alerts and remote notifications through the Blynk IoT app.

## 2. User Perspective

Users interact with SAFESNIFF through the Blynk IoT mobile app, which provides a simple and intuitive interface:

1. Live Dashboard:  
   Displays real-time gas concentration (in PPM). Includes virtual buttons to turn off the alarm (buzzer) and SOS alert (red LED) if the gas level is safe.
2. Alerts and Notifications:  
   When gas concentration exceeds the safety threshold, the system:

* Activates the local alarm (buzzer + red LED)
* Sends a push notification to the user’s phone via Blynk

**Usage Flow:**

1. User installs the Blynk IoT app from Google Play Store or Apple App Store.
2. User **shares their email with the admin** to link their Blynk account to the ESP32 device.
3. Admin provides the user with their **Blynk login credentials.**
4. Once log in, Blynk automatically loads the predefined dashboard with:

* Gas reading gauge
* Visual alert indicator (LED widget)
* Buttons to silence alarm and SOS

1. The app connects to the ESP32 via the Blynk cloud, using Wi-Fi.
2. Once connected:

* Real-time gas readings appear
* System automatically sends notifications if gas crosses threshold
* Users can manually disable alarm/SOS via buttons (only if safe)

## 3. Designer Perspective

### 3.1 System Architecture

Hardware Layer:

* Microcontroller: ESP32
* Sensors: MQ-2 Gas Sensor (for LPG, smoke, methane detection)
* Buzzers/LEDs:
* Piezo buzzer (for local alert)
* LED (for visual warning)
* Connectivity: Wi-Fi module (built-in on ESP32)
* Cloud Service: Blynk Cloud (for remote monitoring and alerts)
* Workflow:
* ESP32 continuously reads analog values from the MQ-2 gas sensor
* If gas level crosses the threshold:  
   – Local buzzer and LED are triggered  
   – ESP32 pushes data to Blynk Cloud via Wi-Fi
* Blynk app displays real-time values and sends push notifications
* Users can remotely silence alarm or trigger SOS via app buttons

Software Layer:

* App: Blynk IoT
* **Communication Protocol:** Blynk's proprietary protocol over Wi-Fi

Features:

* Real-time gas readings displayed via gauge
* Virtual LED for visual alert (gas leak)
* Push notifications triggered on threshold breach
* Button widgets to silence buzzer or send SOS signal

## Functional Requirements

|  |  |
| --- | --- |
| **ID** | **Requirement Description** |
| FR1 | System shall read gas sensor data every 10 seconds. |
| FR2 | System shall display live gas readings on the Blynk dashboard. |
| FR3 | System shall trigger LED widget when gas level exceeds threshold. |
| FR4 | System shall send push notification via Blynk when unsafe levels are detected. |
| FR5 | System shall allow the user to manually reset alarm via app button. |

## 5. Constraints

|  |  |
| --- | --- |
| Category | Constraint Description |
| **Hardware Cost** | Total component cost should stay under PKR 2000–2500. |
| **Power Supply** | Must operate on 5V battery via ESP32. |
| **Latency** | Gas detection response time should be under 2–3 seconds. |
| **Security** | Limited to Blynk’s built-in authentication |
| **Timeline** | The working prototype should be ready within 1 week. |

## 6. Future Enhancements

* Send SMS or call alerts during emergencies.
* Automatically turn on the exhaust fan when gas is detected.

# System Design Document: SAFESNIFF – Real-Time Natural Gas Detection (LPG & Methane) System

## 1. Introduction

This document presents the system design options for **SAFESNIFF**, a real-time natural gas detection project aimed at detecting LPG or methane leaks. The core goal is to capture gas concentration data from sensors and upload it to a live dashboard via Wi-Fi for timely alerts. Design choices have been evaluated based on cost, performance, ease of use, and development timeline.

## 2. System Design Options

### 2.1 ESP32 + MQ2 sensor

The ESP32 is a low-cost microcontroller featuring built-in Wi-Fi, making it a natural fit for this project. Its strong processing power and ample memory allow it to efficiently run multiple tasks simultaneously, from reading sensors and controlling displays to managing alerts, ensuring smooth and reliable operation.

### 2.2 Raspberry Pi

The Raspberry Pi offers higher computational capabilities and built-in Wi-Fi connectivity. However, it comes at a higher price and includes features like an operating system and file system management, which are unnecessary for this project’s scope. Therefore, it is considered more powerful than required (overkill) for the intended purpose.

### 2.3 Arduino UNO

Arduino UNO is a beginner-friendly and budget-conscious microcontroller. It provides enough processing power for sensor management and basic alerts. However, it lacks built-in Wi-Fi, meaning additional hardware is required for wireless connectivity, adding complexity to the system.

## 3. Option Evaluation

Each design option was evaluated based on the following metrics:

|  |
| --- |
| * Cost |

|  |
| --- |
| * Ease of Programming & Development |

|  |
| --- |
| * Community & Library Support |

|  |
| --- |
| * Wi-Fi Integration Capability |

|  |
| --- |
| * Suitability for Real-Time Data Acquisition |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Option** | **Cost** | **Ease of Programming** | **Community and Library Support** | **Wi-Fi integration capability** | **Suitability for Real Time data acquisition** |
| ESP32 | Low | High | Excellent | Built-in | Suitable |
| Raspberry Pi | High | Moderate | Excellent | Built-in | Overkill |
| Arduino UNO | Low | High | Excellent | Requires extra modules | Moderately suitable |

## 4. Final Design Choice

After evaluating all options, the **ESP32** was chosen for the project due to its suitability in terms of cost, usability, and functionality. The ESP32 offers built-in Bluetooth and Wi-Fi features, making it a better fit compared to the Arduino UNO, which requires additional modules, and the Raspberry Pi, which is comparatively expensive. Overall, the ESP32 provides a flexible and cost-effective solution tailored according to the project’s needs.

## 5. Methodology

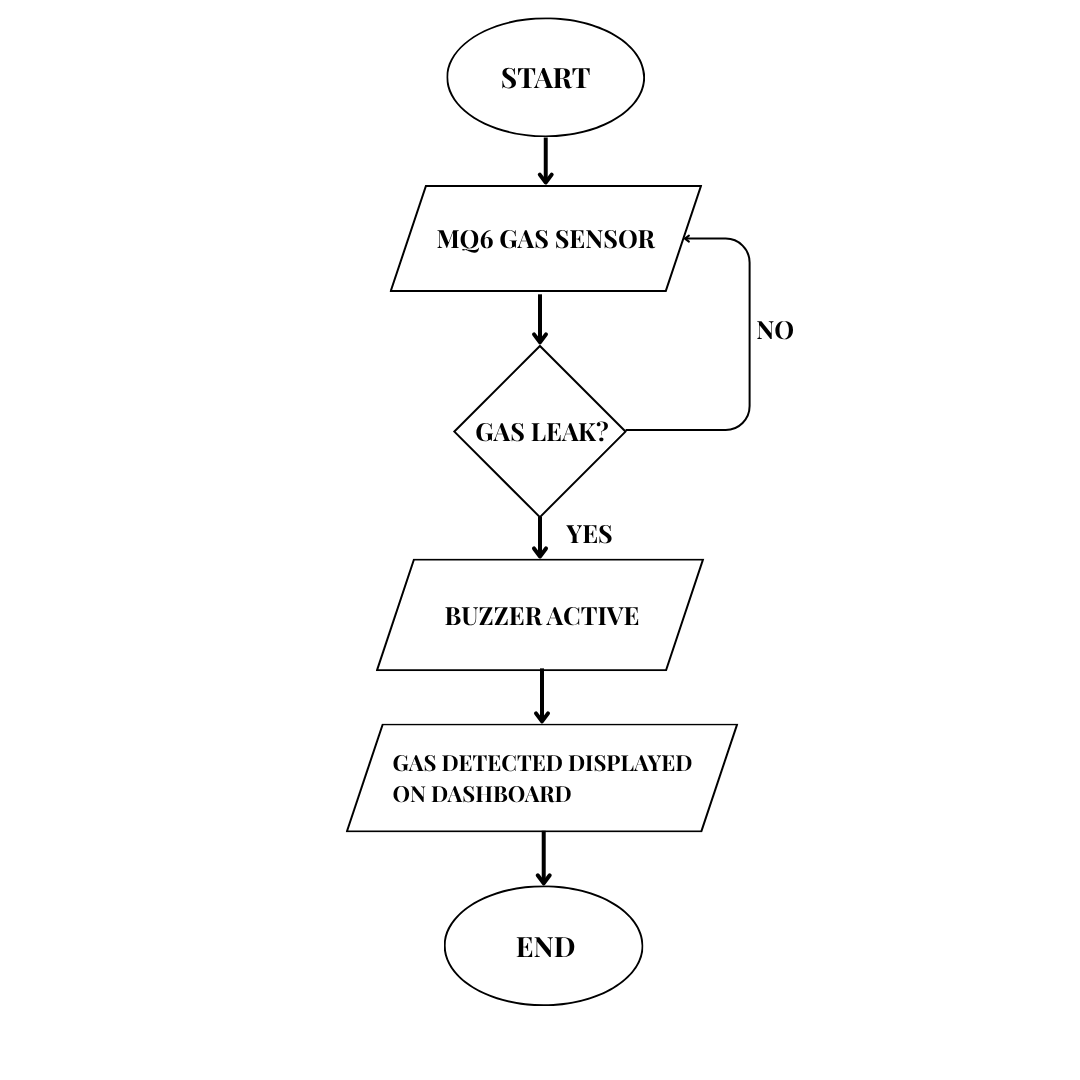


Figure 1: Flow chart

**6. Explanation:**

**6.1 Hardware Approach:**

The functioning mechanism of SAFESNIFF operates on an input-output based structure as illustrated in **Figure 1**. The MQ2 gas sensor serves as the input device, responsible for detecting the presence of combustible gases in the surrounding environment. The ESP32 microcontroller acts as the central processing unit (the "brain") of the system. When the sensor detects gas above a certain threshold, the ESP32 processes this input and activates the appropriate output response. The output in this case is a buzzer, which emits a warning tone to alert nearby individuals of potential danger. This setup ensures immediate, localized notification in the event of a gas leak.

**6.2 Software Approach (Interfacing):**

For real-time monitoring, a dashboard interface has been implemented to visualize the gas concentration levels continuously. This allows users to track the readings dynamically and intervene if gas levels approach hazardous thresholds. The ESP32 is programmed to transmit sensor data to the dashboard with an update interval of 10 milliseconds, ensuring near-instantaneous refresh of values. This high-frequency data update enables a responsive and accurate visualization, supporting proactive safety measures and user awareness.

**1. Introduction**

This document outlines the proposed test procedure for the IoT-based LPG Gas Leakage Detection and Alert System. The system uses an ESP32 microcontroller connected to an MQ-2 gas sensor to detect leakage of LPG, butane, or methane gases. Upon detecting a concentration above a defined threshold (e.g. 300 PPM), it triggers local alerts using a buzzer and LED and remotely notifies the user via push notification using the Blynk IoT platform. The mobile app displays real-time gas concentration and allows manual control of the buzzer and LED.

Testing includes validation of each hardware and software component, followed by integration and functionality testing of the entire system under simulated gas leakage conditions.

**2. Testing of Individual Components**

**2.1 MQ-2 Gas Sensor**

* Connect MQ-2 sensor to ESP32 (analog output to ADC pin).
* Use serial monitor to read real-time voltage values.
* Calibrate sensor using a lighter or butane source to confirm response.
* Check sensor warm-up period (~20 seconds).
* Ensure the analog value increases significantly when exposed to gas.

**2.2 ESP32 WiFi Module**

* Flash ESP32 with a test sketch using Arduino IDE.
* Connect to a known WiFi network.
* Confirm connection via serial monitor output (IP address, status).
* Test HTTP communication by sending a dummy request to Blynk cloud.

**2.3 Buzzer and Red LED**

* Connect the buzzer and red LED to GPIO pins via appropriate resistors.
* Use a test sketch to activate buzzer and LED manually.
* Verify correct activation at expected gas level threshold.
* Confirm both can also be triggered remotely via Blynk app virtual pins.

**2.4 Blynk IoT App (Mobile Interface)**

* Create Blynk template with:
  + Gauge widget (live PPM reading)
  + LED indicator (gas alert status)
  + Two buttons (to turn off buzzer/LED manually)
* Link virtual pins to buzzer/LED logic in code.
* Test mobile notification via logEvent() from ESP32.

**3. Integration Testing**

**3.1 Sensor + ESP32**

* Monitor gas levels in serial plotter.
* Trigger buzzer and LED locally when level exceeds 300 PPM.
* The confirmation system resets when values drop below threshold.

**3.2 ESP32 + Blynk Cloud**

* Ensure ESP32 sends gas data to Blynk every 1–2 seconds.
* Confirm mobile app shows live data on gauge.
* Trigger log Event() on gas threshold breach and confirm push notification on phone.

**3.3 Manual Control from Mobile App**

* From Blynk app, press buttons to toggle LED and buzzer.
* Confirm response time is under 2 seconds.
* Validate Blynk's ability to override alert logic manually.

**3.4 Threshold Adjustment Test**

* Temporarily lower threshold in code (e.g., 200 PPM).
* Confirm that buzzer/LED trigger and notification still work correctly.
* Revert to 300 PPM after test.

**3.5 Noise and False Trigger Testing**

* Expose sensor to smoke or deodorant briefly.
* Ensure system handles it gracefully with no repeated false alarms.

**3.6 Endurance Test**

* Run full system for 2 hours with periodic gas exposure.
* Monitor stability of cloud connectivity, mobile responsiveness, and sensor accuracy.
* Log any disconnections, false triggers, or performance drops.