GAS LEAKAGE MONITORING & ALERTING SYSTEM FOR INDUSTRIES

PROJECT REPORT

TEAM ID	**PNT2022TMID19226**
FACULTY MENTOR	**Ms. KEERTHIKA J**

TEAM MEMBERS:

HARIBOOBAALAN P N
[722819104042]

HARITHAA G
[722819104045]

KARTHIKEYAN C
[722819104054]

KAVIN P
[722819104056]

Title **Page No**

1.INTRODUCTION

1.1 Project Overview

Gas leakage is the main problem of the industrial sectors working with highly inflammable gases. The Gas Leakage Monitoring and Alerting System for Industries actually enables the user to monitor the status of gas leakage in industries 24/7. It also alerts the user and the workers in that industry in case of a gas leakage. The Gas Leakage Monitoring System also sends an SMS notification to the owner of the industry. The system also turns ON the exhaust fans incase of a gas leakage detection, in order to prevent fire accident due to leakage of inflammable gases.

1.2 Purpose

- To provide an IoT device that can detect the gas leakage and perform certain measures to alert people and reduce the impact in case of fire.
- To avoid manned surveillance of industries for gas leakage.
- To identify gas leakage and alert the owner and workers of the industry.
- Turn On exhaust fans to exhaust the inflammable gases to prevent fire accident.

2.LITERATURE SURVEY

2.1 Existing Problem

Gas leakage is the main problem of the industrial sector, residential areas and gas-powered vehicles such as CNG (Compressed Natural Gas) buses and cars. Gas Leakages lead to loss of life and damage to property in case of fire accidents. Therefore, in such cases Industries are in need of a Gas Leakage Detection System for gas leakage identification and to perform certain measures in order to reduce the impact in case of any fire and to alert people/workers about the gas leakage

2.2 Reference

1. Shital Imade, Priyanka Rajmanes, Aishwarya Gavali , Prof. V. N. Nayakwadi “GAS LEAKAGE DETECTION AND SMART ALERTING SYSTEM USING IOT” https://www.pramanaresearch.org/gallery/22.%20feb%20ijirs%20-%20d539.pdf

2. Kumar Keshamoni and Sabbani Hemanth. "Smart Gas Level Monitoring, Booking & Gas Leakage

Detector over IoT " International Advance Computing Conference IEEE, 2017.

3. Petros Spachos , Liang Song and Dimitrios Hatzinakos. "Gas Leak Detection and Localization System Through Wireless Sensor Networks" The 11th Annual IEEE Consumer Communications and Networking Conference - Demos. IEEE, 2014.

4. “Design and Implementation of an Economic Gas Leakage Detector” National Institute of Health (2004). What you need to know about natural gas detectors.

Available: http://www.nidcd.nih.gov/health/smelltaste/gas dtctr.asp.

5. Prof.M.Amsaveni, A.Anurupa, R.S.Anu Preetha, C.Malarvizhi,M.Gunasekaran “Gsm based LPG leakage detection and controlling system” the International Journal of Engineering and Science (IJES) ISSN (e): 2319 – 1813 ISSN (p):2319 – 1805 Pages 112-116 March- 2015

6. Srinivasan,Leela,Jeyabharathi,Kirthika,Rajasree“GAS LEAKAGE DETECTION AND CONTROL” Scientific Journal of Impact Factor(SJIF): 3.134

7. Pal-Stefan Murvaya, IoanSileaa “A survey on gas leak detection and localization techniques”

8. Ch. Manohar Raju, N. Sushma Rani, “An android based automatic gas detection and indication robot. In International Journal of Computer Engineering and Applications. 2014;8(1).

9. Falohun A.S., Oke A.O., Abolaji B.M. “Dangerous Gas Detection using an Integrated Circuit and MQ-9” in International Journal of Computer Applications (0975 –8887) Volume 135 – No.7, February 2016.

10. Ashish Shrivastava,Ratnesh Prabhaker, Rajeev Kumar and Rahul Verma “GSM BASED GAS LEAKAGE DETECTION SYSTEM” in International Journal of Technical Research and Applications e-ISSN: 2320-8163,www.ijtra.com Volume 1, Issue 2 (may-june 2013).

11. C.Selvapriya, S.Sathyaprabha, M.Abdulrahim,” LPG leakage monitoring and multilevel alerting system”, published in 2013.

12. Falohun A.S., Oke A.O., Abolaji B.M. “Dangerous gas detection using an integrated circuit and MQ-9. In International Journal of Computer Applications. 2016; 135(7).

13. Ashish Shrivastava, Ratnesh Prabhaker, Rajeev Kumar, Rahul Verma. GSM based gas leakage detection system. In International Journal of Technical Research and Applications. 2013; 1(2).

2.3 Problem Statement Definition

Gas leakage is the main problem of the industrial sector. Gas Leakage fire took an increasing toll on lives and property in recent years. Most Gases used for industrial activities are highly inflammable and may even catch fire from the source of the leak. The leakage of gases only can be detected by human nearby and if there are no human nearby, it cannot be detected. But sometimes it cannot be detected by human who has a low sense of smell. Therefore, in such cases Industries are in need of a Gas Leakage Detection System for gas leakage identification and to perform certain measures in order to reduce the impact in case of any fire and to alert people/workers about the gas leakage

3.IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas

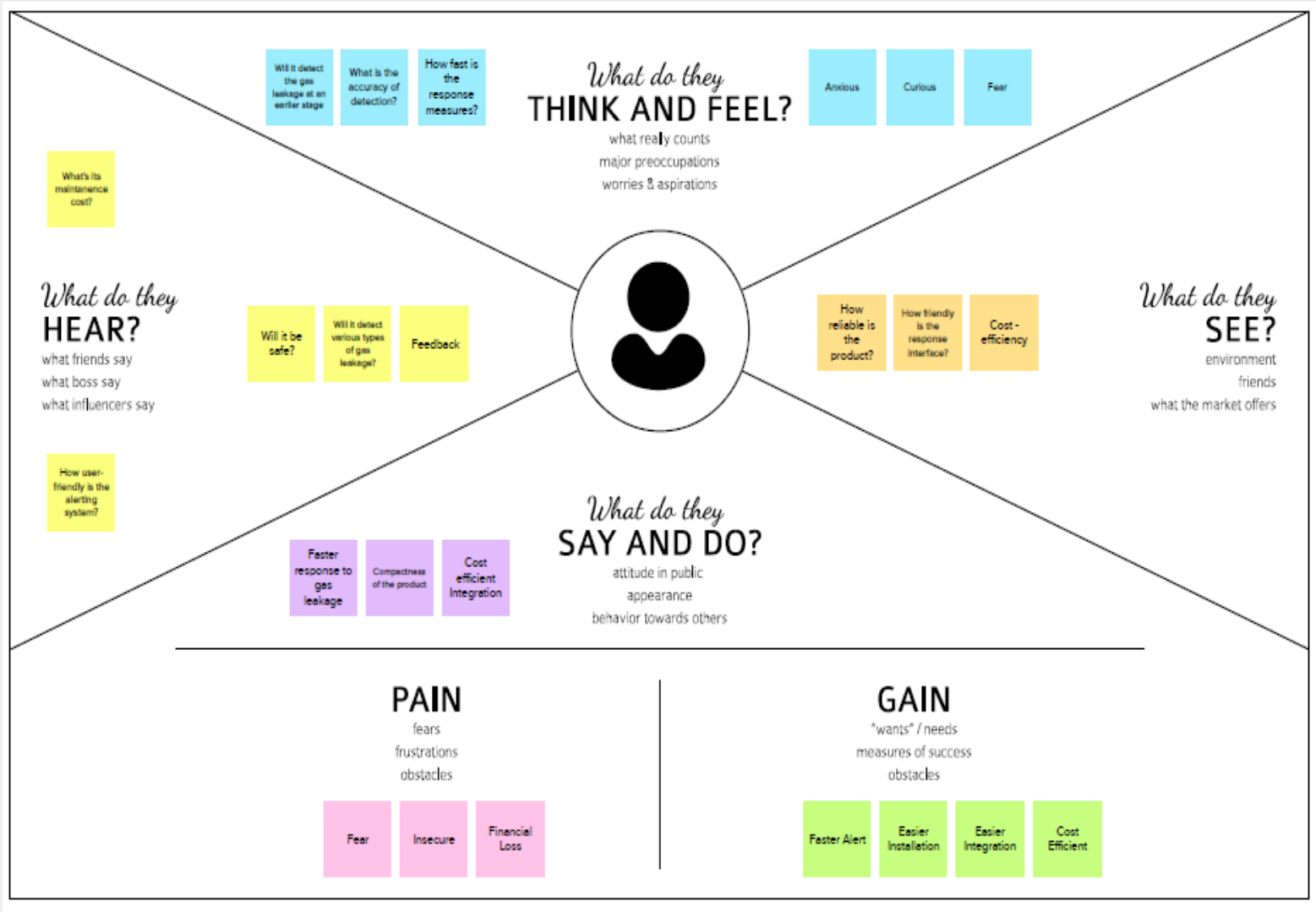


Figure 1: Empathy Map

3.2 Ideation and Brainstorming

<u>Step 1</u>: Team Gathering, Collaboration and Select the Problem Statement

Team was collaborated in mural app.

The team members are

- Haribooobaalan P N
- Harithaa G
- Karthikeyan C
- Kavin P

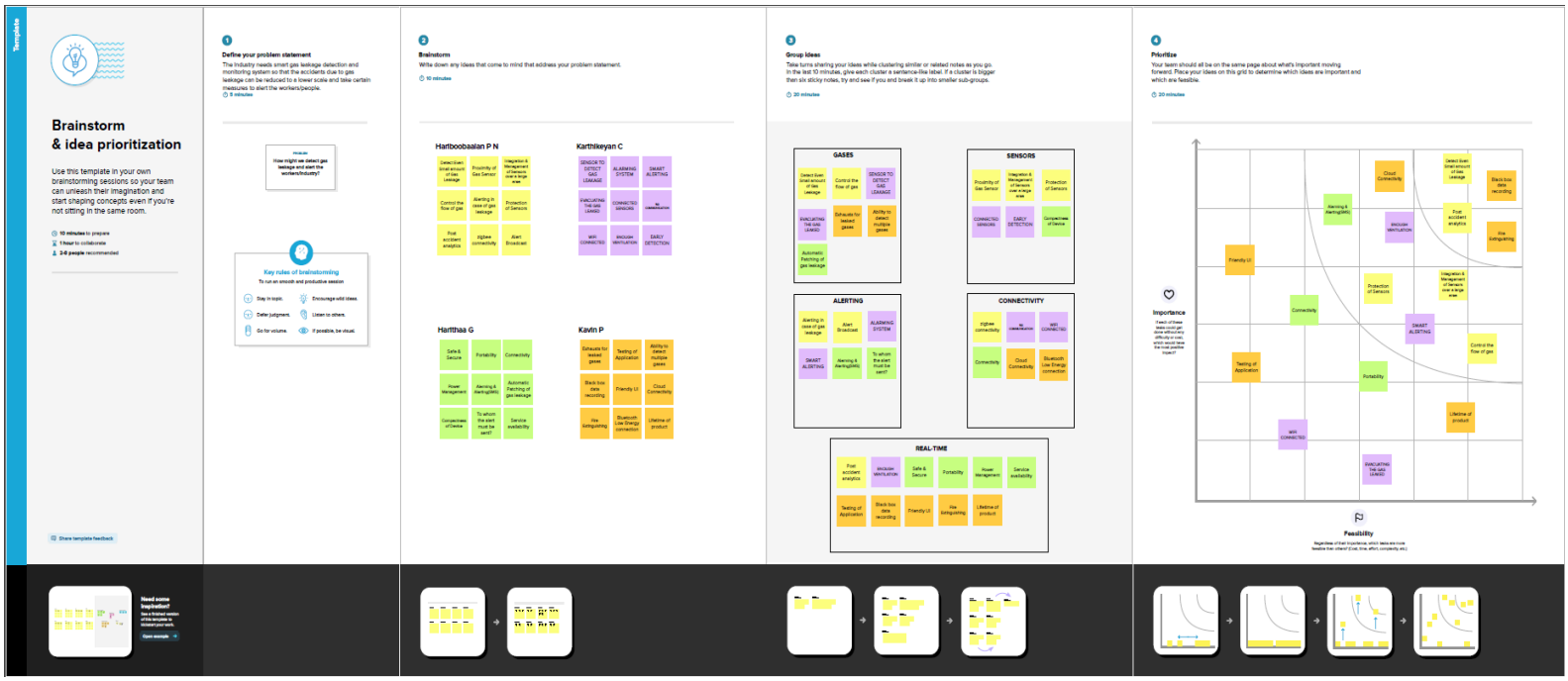


Figure 2: Brainstorming Map

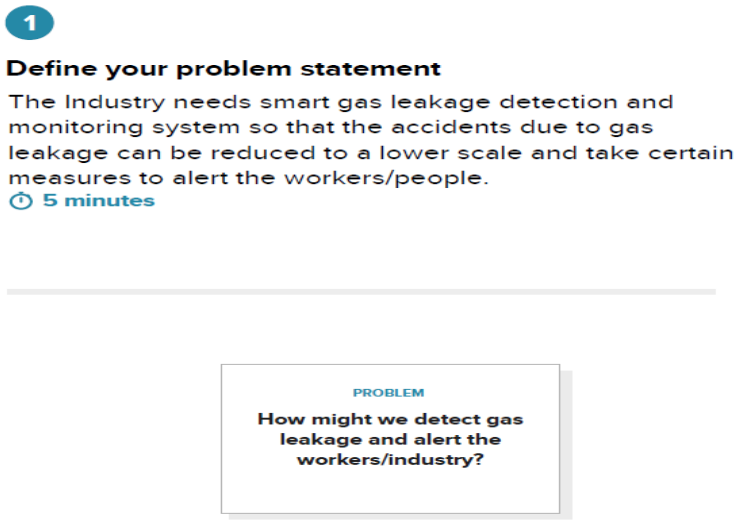


Figure 3: Brainstorming-1

Step 2: Brainstorm, Idea Listing and Grouping

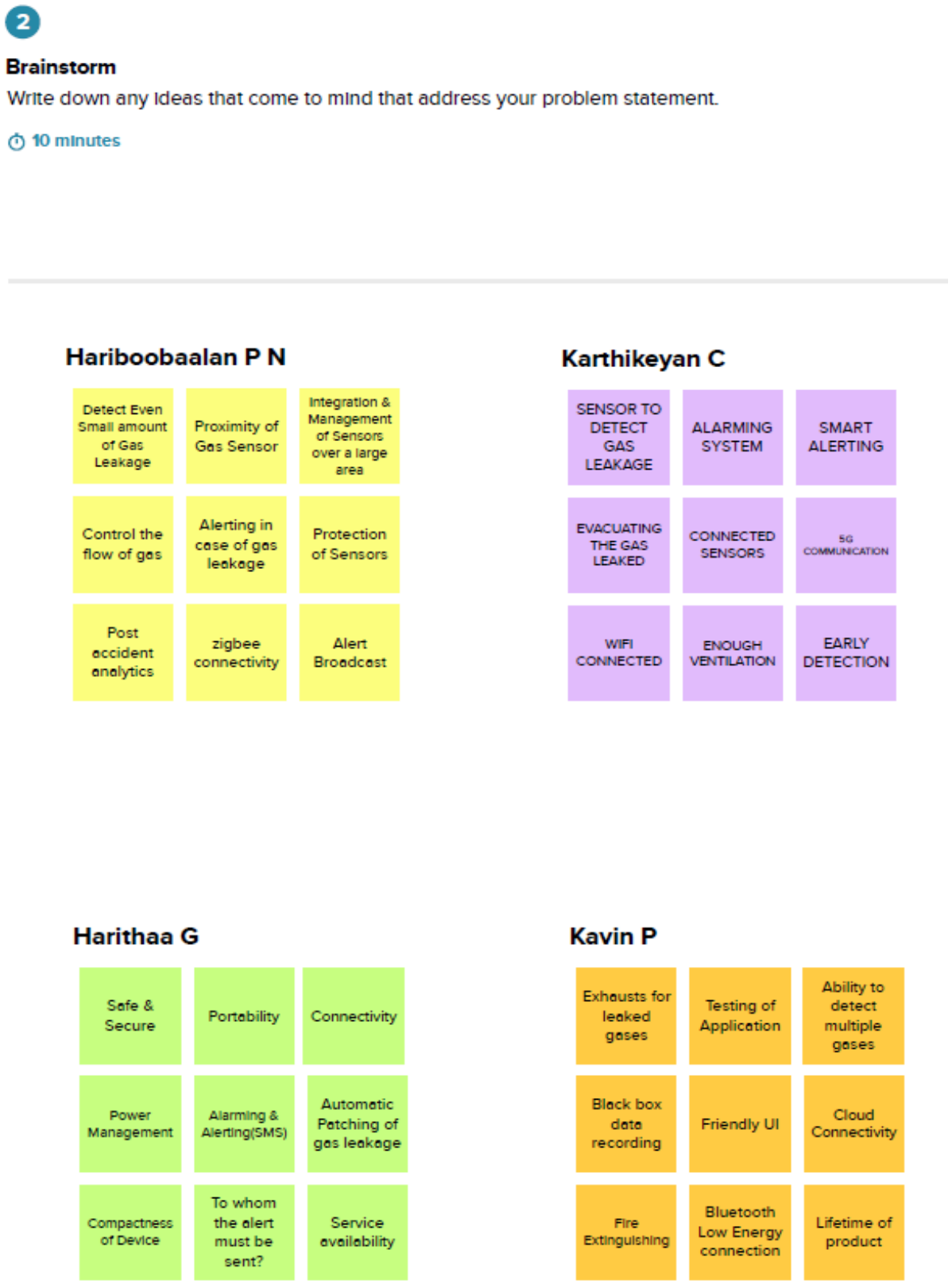


Figure 4: Brainstorming Map-2

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

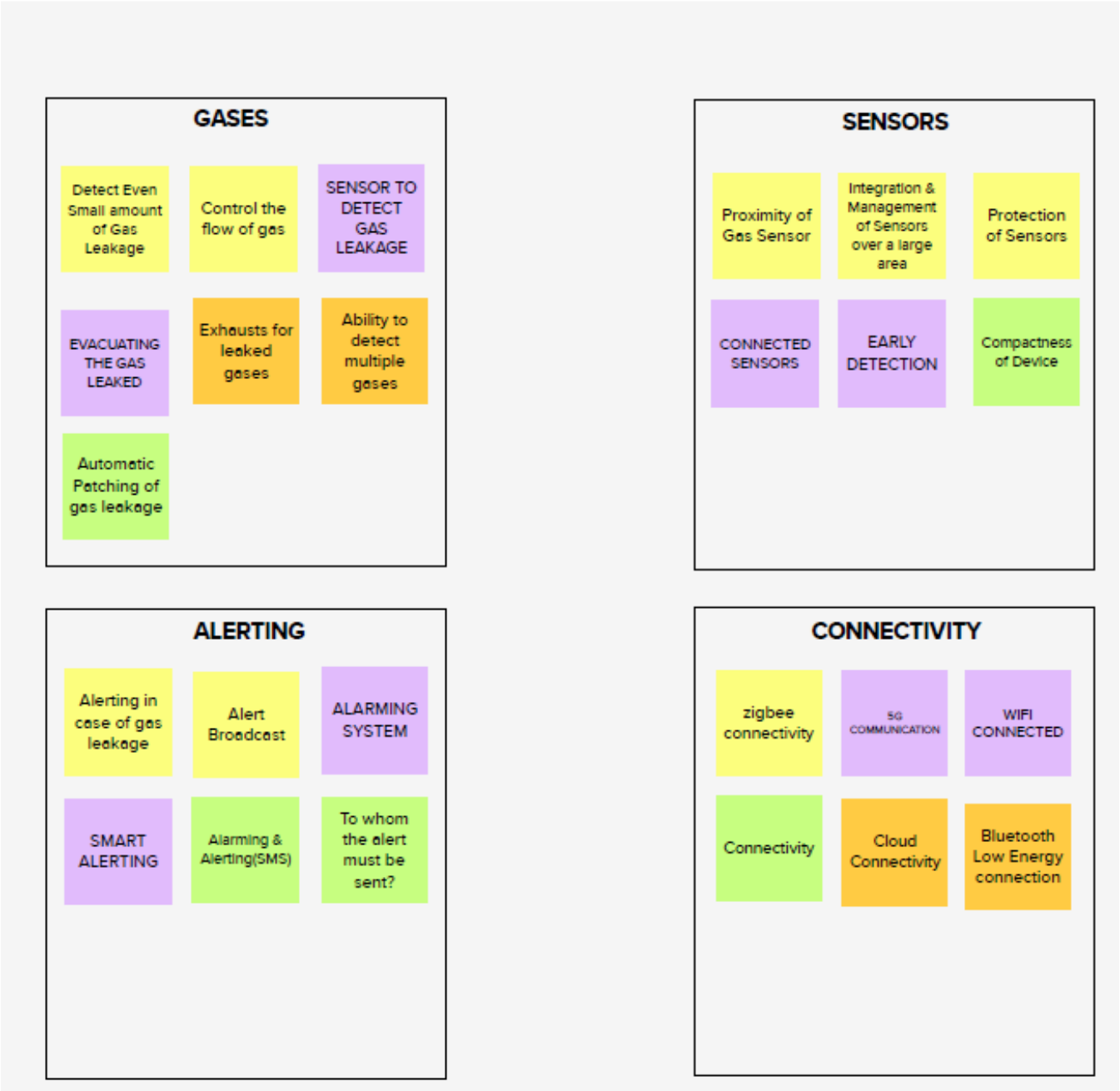


Figure 5: Brainstorming Map 3

Step3: Idea Prioritization

Figure 6: Brainstorming Map-4

3.3 Proposed Solution

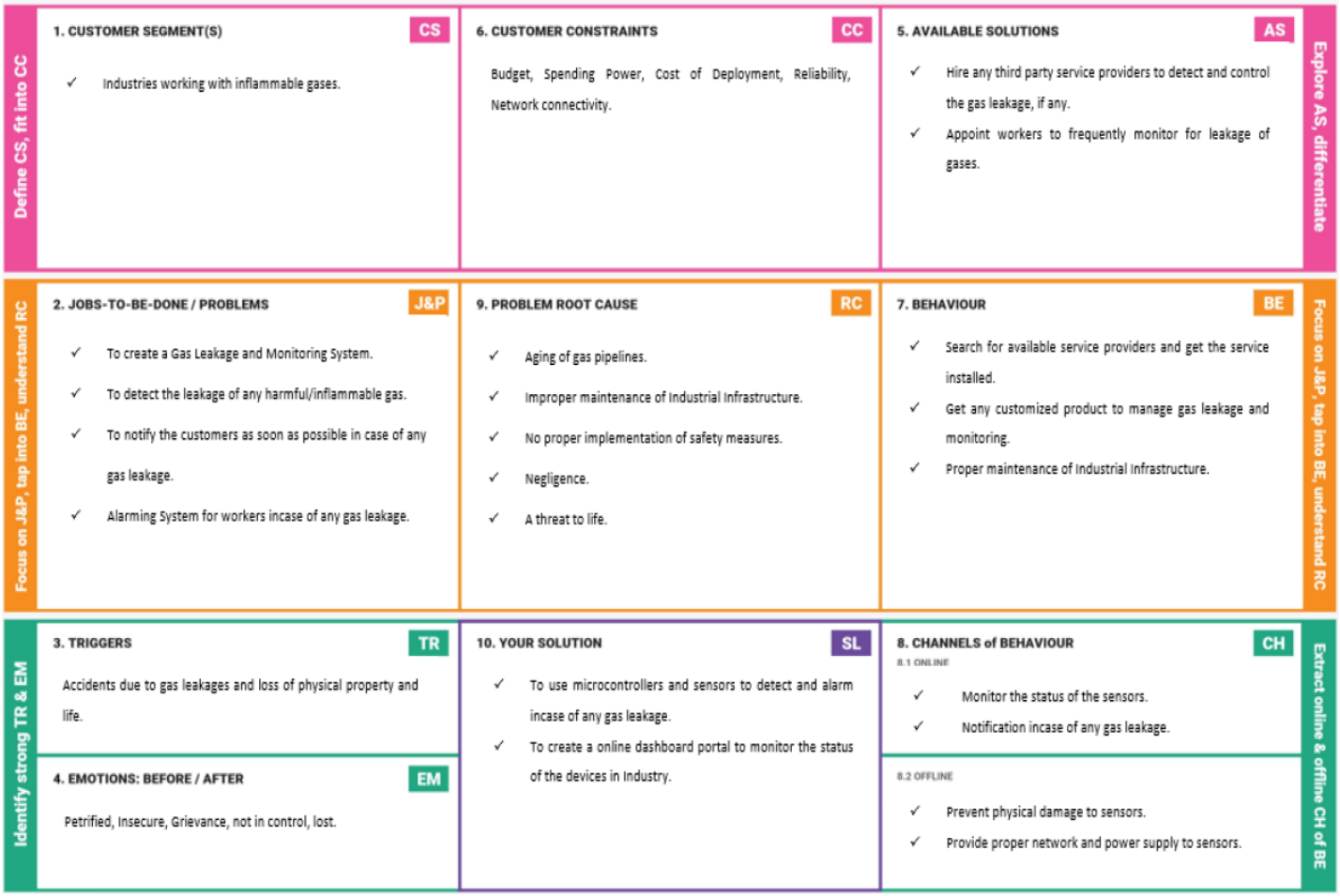
Table 1: Proposed Solution

S No	Parameter	Description
1.	Problem Statement (Problem to be solved)	Gas leakage is the main problem of the industrial sector, residential areas and gas-powered vehicles such as CNG (Compressed Natural Gas) buses and cars. Gas Leakage fires took an increasing toll on lives and property in recent years. Most Gases used for industrial activities are highly inflammable and will even catch fire from the source of the leak. The leakage of gases only may be detected by humans nearby and if there are not any humans nearby, it cannot be detected. But sometimes it cannot be detected by humans who have a coffee sense of smell. Therefore, in such cases Industries are in need of a Gas Leakage Detection System for gas leakage identification and to perform certain measures in order to scale back the impact in case of any fire and to alert people/workers about the gas leakage.
2.	Idea / Solution description	To create a device which periodically monitors the level of the gas in the area of interest and updates the status in the server, which can be viewed using an application. It continuously listens to the level of gas in the atmosphere and provides a warning using a buzzer and provides the alert to the incharge people if any leakage occurs via the GSM network.
3.	Novelty / Uniqueness	Gas leaks cause harm to people's lives and finances. Such leaks have been actively avoided, and effective methods for sensor-based leak detection and localisation have been developed. When these sensors locate a hazardous gas, they often sound an alarm. It alerts the user over the GSM network when a gas leak is detected. The system is made up of an Arduino Uno Microcontroller and a gas leak detection device. The system employs a buzzer to sound.
4.	Social Impact / Customer Satisfaction	Explosions caused by unidentified gas leaks are dangerous for the workers who are exposed to a dangerous atmosphere. For increased safety, it becomes necessary to implement smart systems to precisely identify combustible, flammable, and hazardous gases as well as detect oxygen depletion in industrial buildings. A gas detection system is a fundamental necessity for safety in the oil and gas, hospital, and hotel industries, as well as other settings

		where dangerous gases are frequently employed.
5.	Business Model (Revenue Model)	The product can be made compact, cost efficient and easily installable so that all the industries from small scale to large scale can afford to buy the product which creates more profit. It can even be used for domestic purposes for LPG gases.
6.	Scalability of the Solution	A mobile application can be developed that can provide details about the amount of gas present in the region, set reminders to check gas levels, and anticipate gas leaks by providing values. To increase safety, relay motors can be added to the system. In the event that the gas concentration exceeds a certain threshold, these motors have the ability to turn off the main power and gas supplies.

3.4 Proposed Solution Fit

Figure 7: Proposed Solution Fit



4.REQUIREMENT ANALYSIS

4.1 Functional Requirements

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	➢ Registration through Form ➢ Offline Registration
FR-2	User Confirmation	➢ Confirmation via Email ➢ Confirmation via OTP
FR-3	User Authentication	➢ User verification through valid User ID and password.
FR-4	User Access	➢ Realtime Monitoring of Gas Leakage System, through web portal for Authorized Users.
FR-5	User Alert	➢ User receives an alert through SMS. ➢ Turn on Alerting System in Industry.
FR-6	Review and Feedback	➢ Receive Feedback from Users.

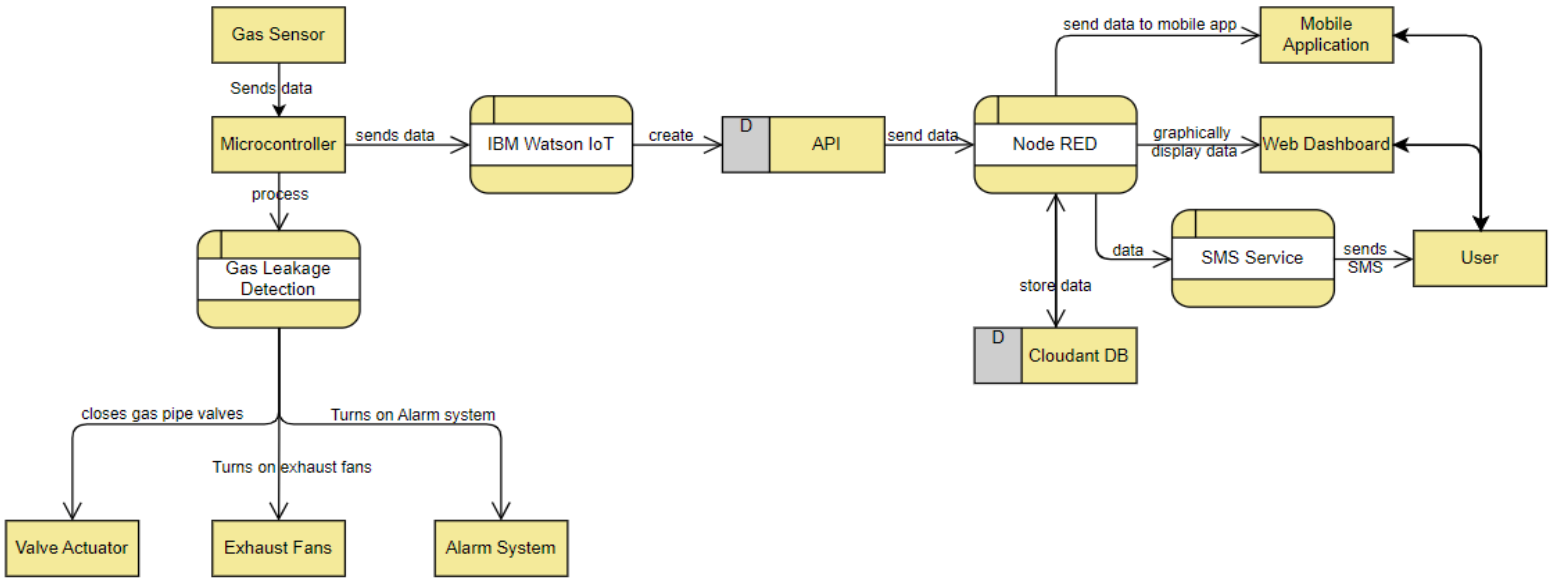
4.2 Non - Functional Requirements

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	➢ Easier Installation process, and Realtime Monitoring Service.
NFR-2	**Security**	➢ Data transmission and handling through secured protocols. ➢ Data encryption & Cloud security.
NFR-3	**Reliability**	➢ Only authorised personnel have access to the system. ➢ Assured Data Security and Information conciseness. ➢ Longer Lifetime of Product/Service.
NFR-4	**Performance**	➢ High Accuracy of gas leakage detection in localized area. ➢ Faster Response to Gas Leakage Detection (SMS alert, valve closing).
NFR-5	**Availability**	➢ The user can access the System 24/7. ➢ Realtime monitoring system.
NFR-6	**Scalability**	➢ The system is scalable even in case of many gas sensors. Or in case of many supervisors.

5.PROJECT DESIGN

5.1 Data flow Diagram

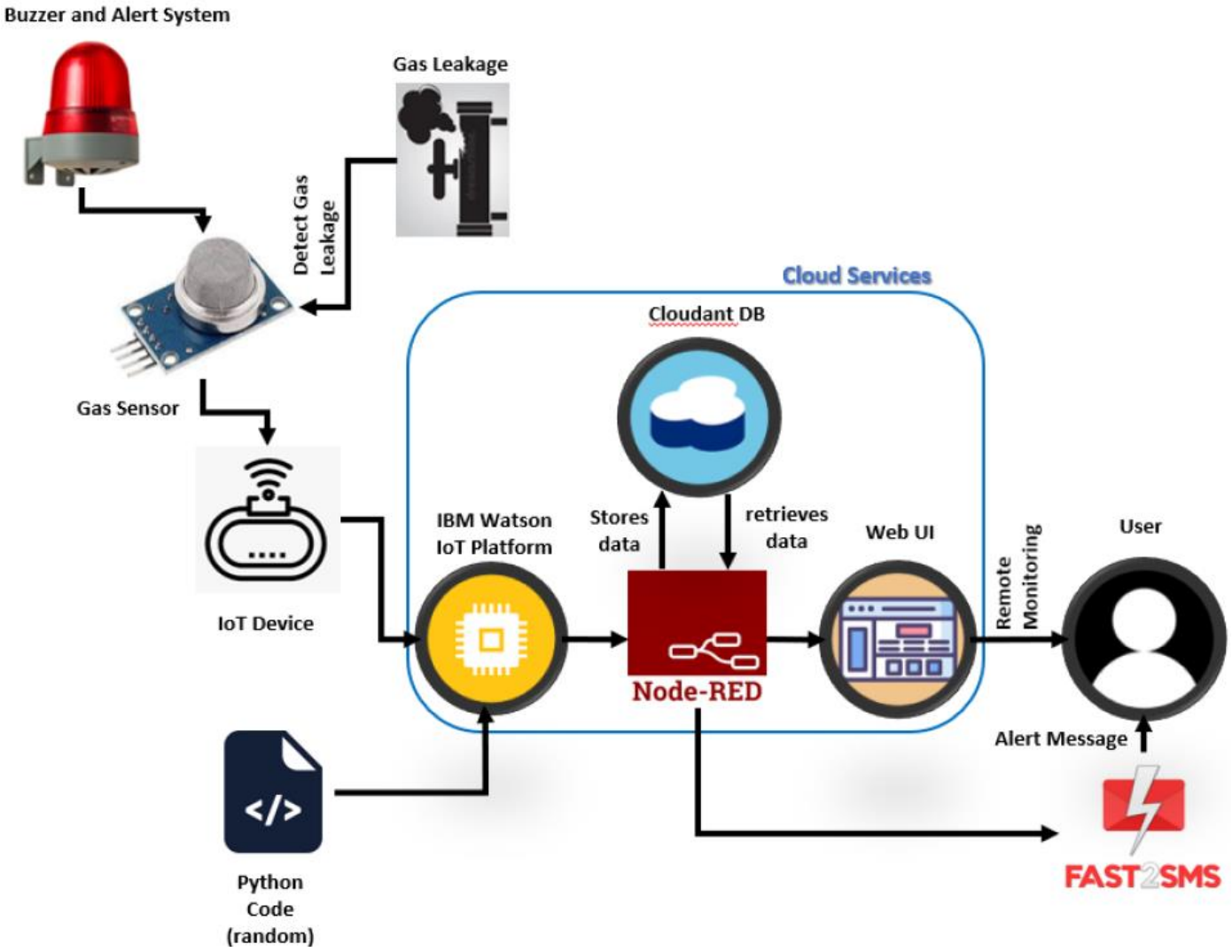
Figure 8: Data Flow Diagram



5.2 Solution and Technical Architecture

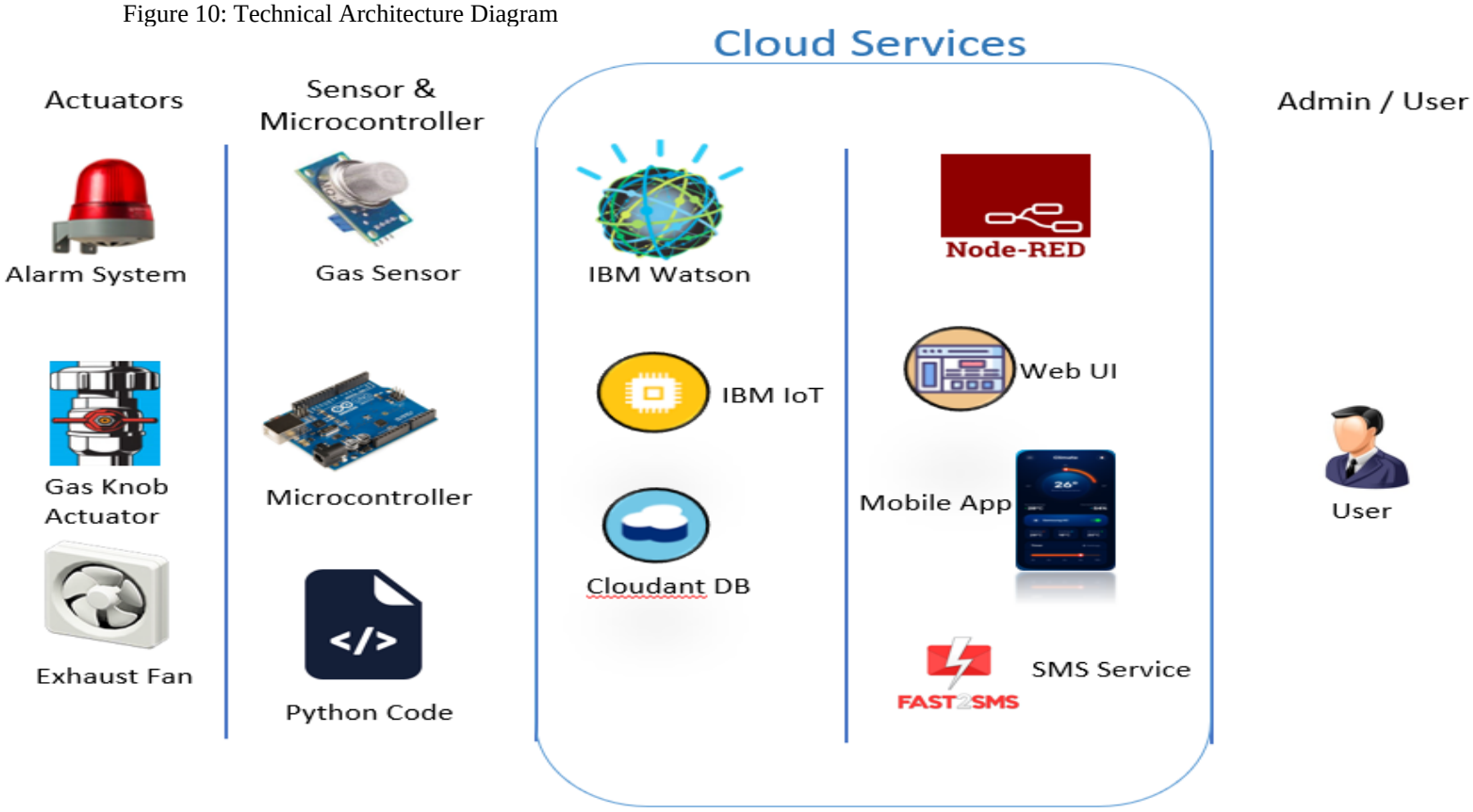
Solution Architecture

Figure 9: Solution architecture



Technical Architecture

Figure 10: Technical Architecture Diagram



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Industry owner)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
Customer (Industry Owner)	Confirmation	USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
Customer (Industry Owner)	Authorize	USN-3	As a user, I will enable the supervisor to monitor the gas leakage system status.	I can provide access to supervisor.	High	Sprint-1
Customer (Supervisor)	Login	USN-4	As a user, I can log into the application by entering email & password.	I can get access to dashboard.	High	Sprint-1
Customer (Supervisor)	Monitor	USN-5	As a user, I can monitor the status of the gas leakage system.	I can view the status of gas leakage system.	High	Sprint-1
Customer (Line Workers)	Notification	USN-6	As a user, I can get (alarm system) alert about gas leakage.	I can get alert about gas leak.	Medium	Sprint-2
Customer (Supervisor)	Notification	USN-7	As a user, I can get SMS notification & alarming alert about gas leakage.	I can get alert about gas leakage.	Medium	Sprint-2
Customer (Industry Owner)	Notification	USN-8	As a user, I can get SMS notification about gas leakage.	I can get alert about gas leakage.	Medium	Sprint-2
Customer (Industry Owner)	Sign-Up	USN-9	As a user, I can sign-up using Facebook login.	I can sign-up with the application using Facebook.	Low	Sprint-3
Customer (Supervisor)	Sign-Up	USN-10	As a user, I can sign-up using Facebook login.	I can sign-up with the application using Facebook.	Low	Sprint-3
Administrator	Service Request	USN-11	As a user, I can request for service in case of any issue with gas leakage monitoring system	I can get service from provider	Low	Sprint-3
Administrator	Increased service	USN-12	As a user, I can request for scaling up the gas leakage monitoring system.	I can get service from the provider.	Low	Sprint-4

6.PROJECT DESIGN AND PLANNING

6.1 Sprint Planning and Estimation

6.2 Sprint Delivery Schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	Low	Karthikeyan C
Sprint-1	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	3	Medium	Hariboobaalan P N
Sprint-1	Registration	USN-3	As a user, I can log into the application by entering email & password.	5	Medium	Kavin P
Sprint-1	Monitoring	USN-4	As a user, I will enable the supervisor to monitor the gas leakage system status.	10	High	Harithaa G
Sprint-2	Notification	USN-5	As a user, I can get SMS notification.	10	High	Harithaa G
Sprint-2	Notification	USN-6	As a user, I can get alarming alert about gas leakage.	10	High	Hariboobaalan P N
Sprint-3	Login	USN-7	As a user, I can sign up using Facebook login	2	Low	Karthikeyan C
Sprint-3	Login	USN-8	As a user, I can sign up using Google Login.	2	Low	Karthikeyan C
Sprint-3	Service Request	USN-9	As a user, I can request for service in case of any issue with the gas leakage monitoring system.	8	Medium	Kavin P
Sprint-3	Service Request	USN-10	As a user, I can request for scaling up the gas leakage monitoring system.	8	Medium	Kavin P
Sprint-4	Monitoring	USN-11	As a user, I can have mobile application for gas leakage monitoring and alerting system.	10	High	Harithaa G
Sprint-4	Monitoring	USN-12	As a user, I must be able to send control commands to initiate measures in case of gas leakage.	10	High	Hariboobaalan P N

6.3 Reports from JIRA

Sprint 1

Figure 11: Sprint-1 Burndown Chart

Sprint 2

Figure 12: Sprint-2 Burndown Chart

Sprint 3
Figure 13: Sprint-3 Burndown Chart

Sprint 4
Figure 14: Sprint-4 Burndown Chart

7.CODING & SOLUTIONING

7.1 Web Application

The web application is used to provide an interface to the user create an account in the Gas Leakage Monitoring System Application. The web application allows the user to create an account in the application by providing User Email, User Password and Mobile Number. And allows the user to log in to their account using their email and password. The application also provides the feature to create an account using Gmail account through Sign in With Google option and to create an account using Facebook account through Sign in With Facebook option. The mobile number entered by the user is used to send the alert message through the Twilio Messaging Service whenever the gas leakage is detected. The logged in user can view the Humidity, Temperature and Gas Level through in embedded Node-RED Dashboard.

File Structure:

Figure 15: File Structure

Code Split Up:

index.html

It is the main page of the web application.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <title>PNT2022TMID19226</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

index.js

It is the main component of the application.

```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App"

import "./index.css"

const root = ReactDOM.createRoot(document.getElementById("root"))
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
)
```

index.css

It is used to set overall styling for the application.

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto", "Oxygen",
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, "Courier New",
    monospace;
}
```

app.js

It has the routes for the components and pages to be routed, it contains the authentication provider, protected routes, and other components of the web application.

```
import { Routes, Route, BrowserRouter as Router } from "react-router-dom"
import { UserAuthContextProvider } from "./context/UserAuthContext"
import ProtectedRoute from "./components/protected/ProtectedRoute"
import SignUp from "./components/auth/Signup"
import Login from "./components/auth/Login"
import VerifyEmail from "./components/auth/VerifyEmail"
import SignOut from "./components/auth/SignOut"
import Dashboard from "./components/Dashboard"

import "bootstrap/dist/css/bootstrap.min.css"
import "./css/styles.css"
import ServiceRequest from "./components/other/ServiceRequest"
import Requested from "./components/other/Requested"

function App() {
  return (
    <Router>
      <UserAuthContextProvider>
        <SignOut />
        <Routes>
          <Route exact path="/" element={<Login />} />
          <Route exact path="/signup" element={<SignUp />} />
          <Route exact path="/verifyemail" element={<VerifyEmail />} />
          <Route element={<ProtectedRoute />}>
            <Route path="/dashboard" element={<Dashboard />} />
            <Route exact path="/service_request" element={<ServiceRequest />} />
            <Route exact path="/requested" element={<Requested />} />
          </Route>
        </Routes>
      </UserAuthContextProvider>
    </Router>
  )
}

export default App
```

style.css

It is the main style sheet for the application, which contains the styling for the pages and components in the application.

```
*,
*:before,
*:after {
  padding: 0;
  margin: 0;
  box-sizing: border-box;
}
.main {
  height: 100vh;
  background-color: #080710;
}
.background {
  width: 430px;
  height: 520px;
  position: absolute;
  transform: translate(-50%, -50%);
  left: 50%;
  top: 50%;
}
.background .shape {
  height: 200px;
  width: 200px;
  position: absolute;
  border-radius: 50%;
}
.shape:first-child {
  background: linear-gradient(#1845ad, #23a2f6);
  left: -80px;
  top: -80px;
}
```

```css
.shape:last-child {
  background: linear-gradient(to right, #ff512f, #f09819);
  right: -30px;
  bottom: -80px;
}
.form {
  width: 400px;
  min-height: 60vh;
  background-color: rgba(255, 255, 255, 0.13);
  position: absolute;
  transform: translate(-50%, -50%);
  top: 50%;
  left: 50%;
  border-radius: 10px;
  backdrop-filter: blur(10px);
  border: 2px solid rgba(255, 255, 255, 0.1);
  box-shadow: 0 0 40px rgba(8, 7, 16, 0.6);
  padding: 30px;
}
.form * {
  font-family: "Poppins", sans-serif;
  color: #ffffff;
  letter-spacing: 0.5px;
  outline: none;
  border: none;
}
.form h3 {
  font-size: 32px;
  font-weight: 500;
  line-height: 42px;
```

```
  text-align: center;
}

label {
  display: block;
  margin-top: 30px;
  font-size: 16px;
  font-weight: 500;
}
input {
  display: block;
  height: 50px;
  width: 100%;
  background-color: rgba(255, 255, 255, 0.07);
  border-radius: 3px;
  padding: 0 10px;
  margin-top: 8px;
  font-size: 14px;
  font-weight: 300;
}
::placeholder {
  color: #e5e5e5;
}
.button {
  margin-top: 40px;
  width: 100%;
  background-color: #ffffff;
  color: #080710;
  padding: 10px 0;
  font-size: 18px;
```

```
  font-weight: 600;
  border-radius: 5px;
  cursor: pointer;
}
.social {
  margin-top: 30px;
  display: flex;
}
.social div {
  background: red;
  width: 150px;
  border-radius: 3px;
  padding: 5px 10px 10px 5px;
  background-color: rgba(255, 255, 255, 0.27);
  color: #eaf0fb;
  text-align: center;
}
.social div:hover {
  background-color: rgba(255, 255, 255, 0.47);
}
.social .fb {
  margin-left: 25px;
}
.social i {
  margin-right: 4px;
}
.signout {
  position: fixed;
  right: 0;
  padding: 10px 20px;
```

```
  font-weight: 900;
  margin: 1vh 1vw;
  color: white;
  border-radius: 5px;
  background-color: rgb(202, 56, 56);
  cursor: pointer;
  z-index: 10;
}
.signout:hover {
  background-color: red;
}
.request {
  position: fixed;
  right: 0;
  padding: 10px 20px;
  font-weight: 900;
  margin: 1vh 8rem;
  color: white;
  border-radius: 5px;
  background-color: rgb(56, 56, 202);
  cursor: pointer;
  z-index: 10;
}
.request:hover {
  background-color: blue;
}
.hidden {
  visibility: hidden;
}
.social-button {
```

```
  background-color: rgb(66, 133, 244);
  color: rgb(255, 255, 255);
  height: 50px;
  width: 240px;
  border: none;
  text-align: center;
  box-shadow: rgba(0, 0, 0, 0.25) 0px 2px 4px 0px;
  font-size: 16px;
  line-height: 48px;
  display: block;
  border-radius: 1px;
  transition: background-color 0.218s ease 0s, border-color 0.218s ease 0s,
    box-shadow 0.218s ease 0s;
  font-family: Roboto, arial, sans-serif;
  cursor: pointer;
  user-select: none;
}
.social-button:hover {
  box-shadow: rgba(66, 133, 244, 0.75) 0px 0px 1px 1px;
}

.social-button-svg {
  width: 48px;
  height: 48px;
  text-align: center;
  display: block;
  margin-top: 1px;
  margin-left: 1px;
  float: left;
  background-color: rgb(255, 255, 255);
  border-radius: 1px;
  white-space: nowrap;
}
```

userAuthContext.js

It contains the authentication and user creation codes. It has the features like signup, login, sign in with Google, sign in with Facebook, sign out features. It contains the feature to create the user and update the information of the user in the database.

```
import { createContext, useContext, useEffect, useState } from "react"
import {
  auth,
  googleAuthProvider,
  facebookAuthProvider,
  db,
} from "../firebase/firebaseConfig"

import { ref, child, get, set } from "firebase/database"
import {
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut,
  onAuthStateChanged,
  sendEmailVerification,
  signInWithPopup,
} from "firebase/auth"

const userAuthContext = createContext()
export const UserAuthContextProvider = ({ children }) => {
  const [user, setUser] = useState("")

  const addUser = async (userDetails) => {
    // Add User
    const key = userDetails.email.replaceAll(".", "_")
    get(child(ref(db, "users"), key)).then((snapshot) => {
      if (snapshot.exists()) {
        alert("User Exists")
      } else {
        const nodeRef = child(ref(db, "users"), key)
```

```
      set(nodeRef, { ...userDetails })
      setCurrentUser(userDetails.email)
    }
  })
}
const setCurrentUser = async (email) => {
  const key = email.replaceAll(".", "_")
  get(child(ref(db, "users"), key)).then((snapshot) => {
    if (!snapshot.exists()) {
      alert("User Not Exists")
    } else {
      const nodeRef = ref(db, "currentUser")
      const phone = snapshot.val().mobile
      set(nodeRef, { phone })
    }
  })
}

const signUp = (email, password, mobile) => {
  return createUserWithEmailAndPassword(auth, email, password).then(
    (user) => {
      if (user.user && user.user.emailVerified === false) {
        sendEmailVerification(user.user)
        console.log("Verification Email Sent")
      }
      addUser({ email, mobile })
    }
  )
}
```

```
const signInWithGoogle = async () => {
  await signInWithPopup(auth, googleAuthProvider)
}

const signInWithFacebook = async () => {
  await signInWithPopup(auth, facebookAuthProvider).then((user) => {
    if (user.user && user.user.emailVerified === false) {
      sendEmailVerification(user.user)
      console.log("Verification Email Sent")
    }
  })
}

const login = (email, password) => {
  setCurrentUser(email)
  return signInWithEmailAndPassword(auth, email, password)
}

const signOutGoogle = async () => {
  await signOut(auth)
}

const sendVerificationEmail = () => {
  if (user && user.emailVerified === false) {
    sendEmailVerification(user)
    console.log("Verification Email Sent")
  }
}

useEffect(() => {
```

```
    const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
      setUser(currentUser)
    })
    return () => {
      unsubscribe()
    }
  }, [])
  return (
    <userAuthContext.Provider
      value={{
        user,
        signUp,
        login,
        signOut: signOutGoogle,
        signInWithGoogle,
        signInWithFacebook,
        sendVerificationEmail,
      }}
    >
      {children}
    </userAuthContext.Provider>
  )
}
export const useUserAuth = () => {
  return useContext(userAuthContext)
}
```

Dashboard.js

It is the main component with displays the embedded Node-RED Dashboard of the logged in user.

```
const Dashboard = () => {
  return (
    <div>
      <iframe
        referrerPolicy="no-referrer"
        title="Node-RED Dashboard"
        style={{
          width: "100%",
          height: "100%",
          overflowY: "hidden",
          top: 0,
          left: 0,
          position: "absolute",
        }}
        src="https://node-red-feuln-2022-10-08.au-syd.mybluemix.net/ui/"
      ></iframe>
    </div>
  )
}

export default Dashboard
```

ProtectedRoute.js

It is used to create protected route which allows only verified users to access the protected pages.

```
import { Outlet, Navigate } from "react-router-dom"
import { useUserAuth } from "../../context/UserAuthContext"

const ProtectedRoute = () => {
  const { user } = useUserAuth()
  return user ? (
    user?.emailVerified ? (
      <Outlet />
    ) : (
      <Navigate to="./verifyemail" />
    )
  ) : (
    <Navigate to="./" />
  )
}

export default ProtectedRoute
```

Signup.js

It contains the code for the signup feature.

```jsx
import React, { useState } from "react"
import { Link, useNavigate } from "react-router-dom"
import { Alert } from "react-bootstrap"
import GoogleButton from "react-google-button"
import { useUserAuth } from "../../context/UserAuthContext"

import facebookIcon from "../../images/facebook.png"

const Signup = () => {
  const [email, setEmail] = useState("")
  const [error, setError] = useState("")
  const [password, setPassword] = useState("")
  const [mobile, setMobile] = useState("")
  const { user, signUp, signInWithGoogle, signInWithFacebook } = useUserAuth()
  const navigate = useNavigate()
  const createUser = async (e) => {
    e.preventDefault()
    setError("")
    try {
      await signUp(email, password, mobile)
    } catch (err) {
      setError(err.message)
    }
  }
  const checkUserCreated = () => {
    if (user) navigate("/dashboard")
  }
  return (
    <div className="main" onLoad={checkUserCreated()}>
      <div className="background">
```

```
  <div className="shape"></div>
  <div className="shape"></div>
</div>
<form className="form" onSubmit={createUser}>
  <h3>SignUp Here</h3>
  {error && <Alert variant="danger">{error}</Alert>}

  <label htmlFor="username">Username</label>
  <input
    type="text"
    placeholder="Email or Phone"
    id="username"
    required
    onChange={(e) => setEmail(e.target.value)}
  />

  <label htmlFor="password">Password</label>
  <input
    type="password"
    placeholder="Password"
    id="password"
    required
    onChange={(e) => setPassword(e.target.value)}
  />

  <label htmlFor="phone">Mobile Number</label>
  <input
    type="tel"
    placeholder="Enter 10 digit Mobile Number"
    id="phone"
```

```
            pattern="[0-9]{10}"
            required
            onChange={(e) => setMobile(e.target.value)}
          />

          <button className="button">Sign Up</button>
          <GoogleButton className="m-auto my-3" onClick={signInWithGoogle} />
          <button
            className="m-auto my-3 social-button"
            type="button"
            onClick={signInWithFacebook}
          >
            <div className="social-button-svg">
              <img src={facebookIcon} width="48px" />
            </div>
            <span>Sign in with Facebook</span>
          </button>
          <div className="text-center p-1 m-auto m-1 ">
            Already have an account? <Link to="/">Log In</Link>
          </div>
        </form>
      </div>
    )
  }

  export default Signup
```

Login.js

It contains the code for the login feature.

```jsx
import React, { useState, useEffect } from "react"
import { Link, useNavigate } from "react-router-dom"
import { Alert } from "react-bootstrap"
import GoogleButton from "react-google-button"
import { useUserAuth } from "../../context/UserAuthContext"

import facebookIcon from "../../images/facebook.png"

const Login = () => {
  const [email, setEmail] = useState("")
  const [error, setError] = useState("")
  const [password, setPassword] = useState("")
  const { user, login, signInWithGoogle, signInWithFacebook } = useUserAuth()
  const navigate = useNavigate()
  useEffect(() => {
    if (user) navigate("/dashboard")
  }, [])

  const loginUser = async (e) => {
    e.preventDefault()
    setError("")
    try {
      await login(email, password)
      navigate("/dashboard")
    } catch (err) {
      setError(err.message)
    }
  }
  const checkLogedIn = () => {
    if (user) navigate("/dashboard")
```

```
}
return (
  <div className="main" onLoad={checkLogedIn()}>
    <div className="background">
      <div className="shape"></div>
      <div className="shape"></div>
    </div>
    <form className="form" onSubmit={loginUser}>
      <h3>Login Here</h3>
      {error && <Alert variant="danger">{error}</Alert>}

      <label htmlFor="username">Username</label>
      <input
        type="text"
        placeholder="Email or Phone"
        id="username"
        onChange={(e) => setEmail(e.target.value)}
      />

      <label htmlFor="password">Password</label>
      <input
        type="password"
        placeholder="Password"
        id="password"
        onChange={(e) => setPassword(e.target.value)}
      />

      <button className="button">Log In</button>
      <GoogleButton className="m-auto my-3" onClick={signInWithGoogle} />
      <button
```

```
            className="m-auto my-3 social-button"
            type="button"
            onClick={signInWithFacebook}
          >
            <div className="social-button-svg">
              <img src={facebookIcon} width="48px" />
            </div>
            <span>Sign in with Facebook</span>
          </button>

          <div className="text-center p-1 m-auto m-1 ">
            Don't have an account? <Link to="/signup">Sign Up</Link>
          </div>
        </form>
      </div>
    )
  }

  export default Login
```

SignOut.js

It provides the sign out feature.

```
import { useNavigate } from "react-router-dom"
import { useUserAuth } from "../../context/UserAuthContext"

const SignOut = () => {
  const navigate = useNavigate()
  const { user, signOut } = useUserAuth()
  return (
    <>
      <div className={`signout ${!user && "hidden"}`} onClick={signOut}>
        SignOut
      </div>
      <div
        className={`request ${!user && "hidden"}`}
        onClick={() => navigate("/service_request")}
      >
        Service
      </div>
    </>
  )
}

export default SignOut
```

VerifyEmail.js

It contains the verify email page.

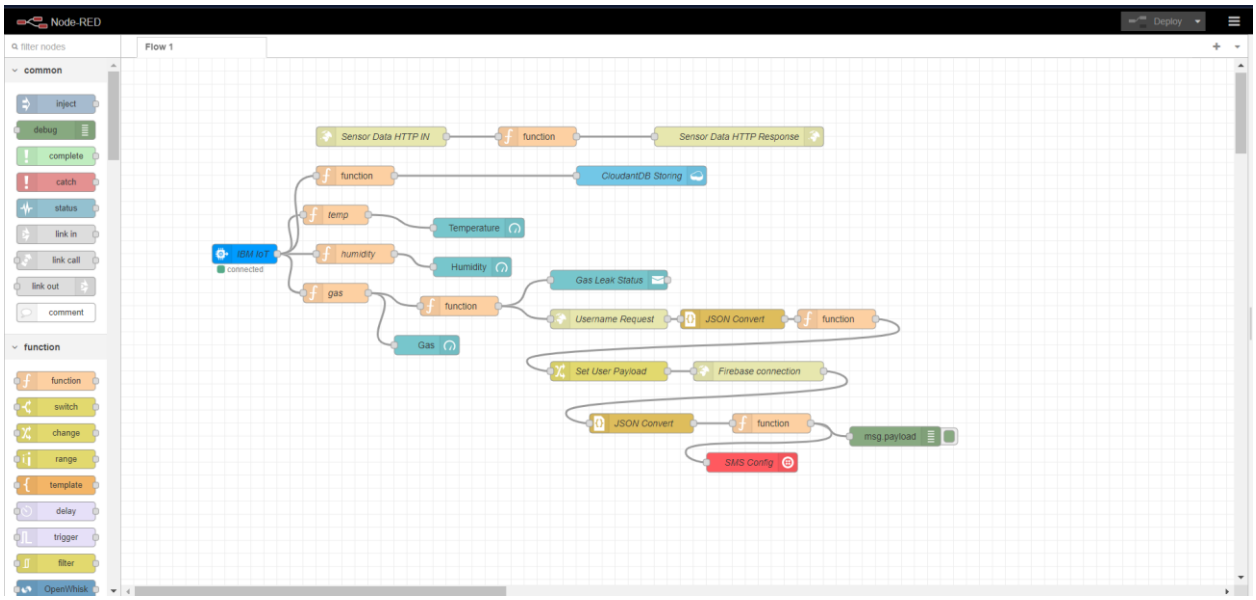
```
import { useNavigate } from "react-router-dom"
// import GoogleButton from "react-google-button"
import { useUserAuth } from "../../context/UserAuthContext"

const VerifyEmail = () => {
  const navigate = useNavigate()
  const { user, sendVerificationEmail } = useUserAuth()

  const checkEmailVerified = () => {
    if (!user) navigate("/")
    if (user && user.emailVerified) {
      navigate("/dashboard")
    }
  }
  const refresh = () => {
    window.location.reload(false)
  }
  return (
    <div className="main" onLoad={checkEmailVerified()}>
      <div className="background">
        <div className="shape"></div>
        <div className="shape"></div>
      </div>
      <div className="form ">
        <div
          style={{
            textAlign: "center",
            veritcalAlign: "center",
            paddingTop: "50%",
            color: "white",
```

```
            fontWeight: "900",
            fontSize: "2rem",
          }}
        >
          <p>
            Verify Your Email
            <br />
            Check Your Inbox!
            <br />
            Check Spam Too!
          </p>
          <div className="button" onClick={refresh}>
            Refresh
          </div>
          <div className="button" onClick={() => sendVerificationEmail()}>
            Resend Verification Email
          </div>
        </div>
      </div>
    </div>
  )
}

export default VerifyEmail
```

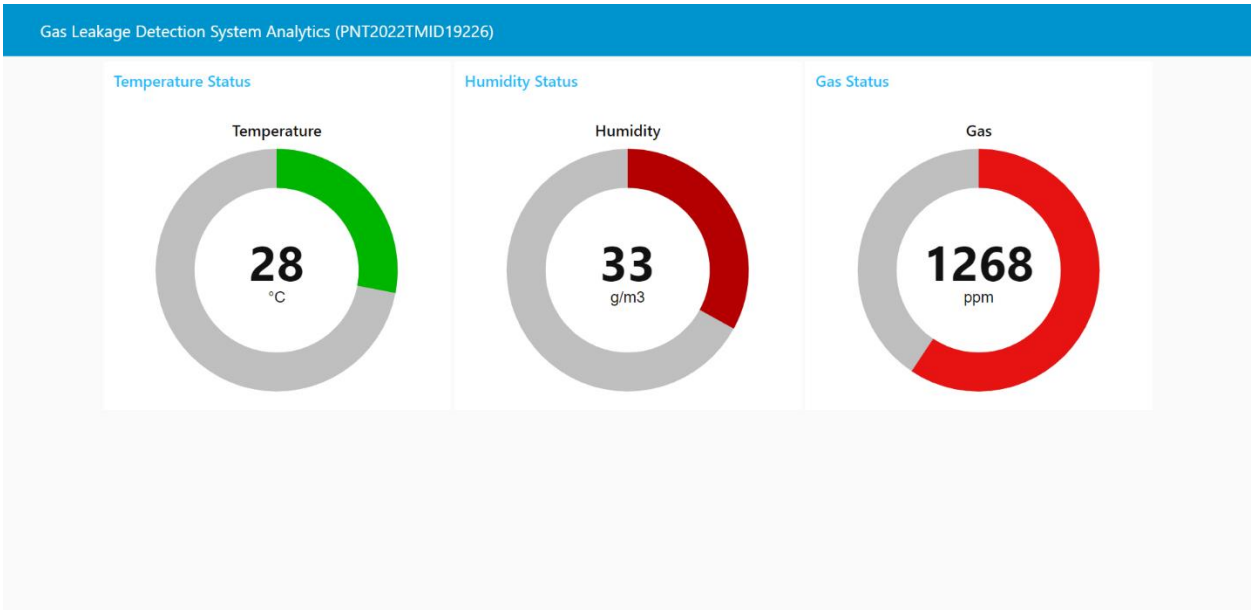
7.1 Node-RED Web Dashboard

The Node-RED dashboard displays the Humidity, Temperature and Gas Level of the Gas Leakage Monitoring System. The Node-RED Dashboard is configured in the Node-RED.

Figure 16: NodeRED Configuration



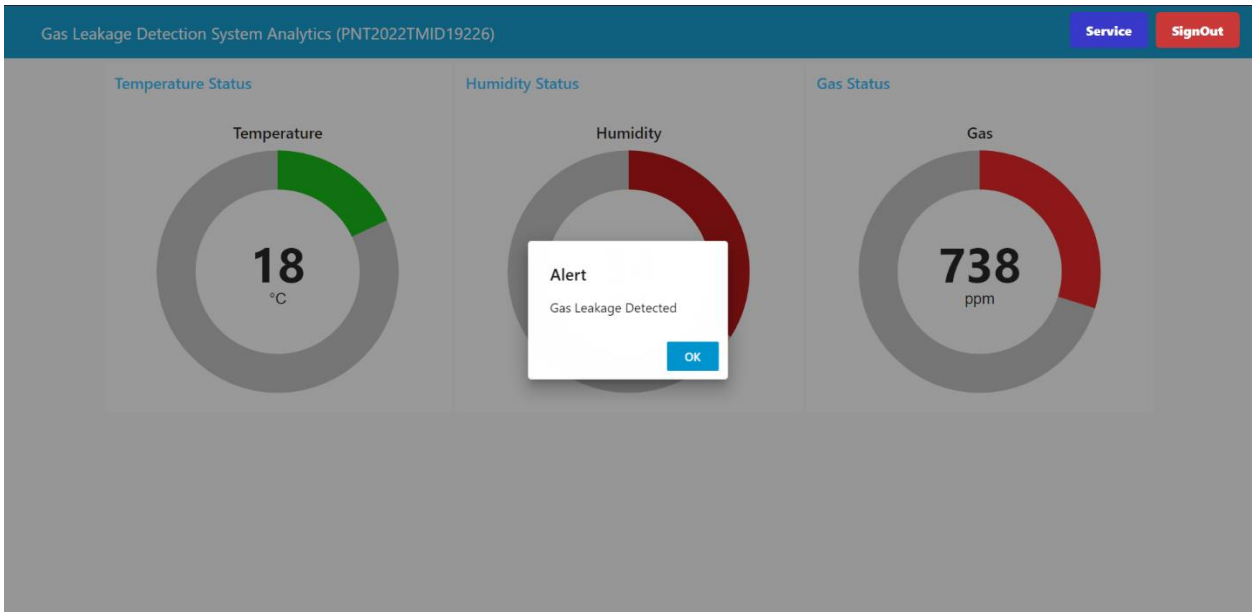
Node RED Configuration



Node RED Dashboard

Figure 17: NodeRED Dashboard

Figure 18: NodeRED Dashboard Alert

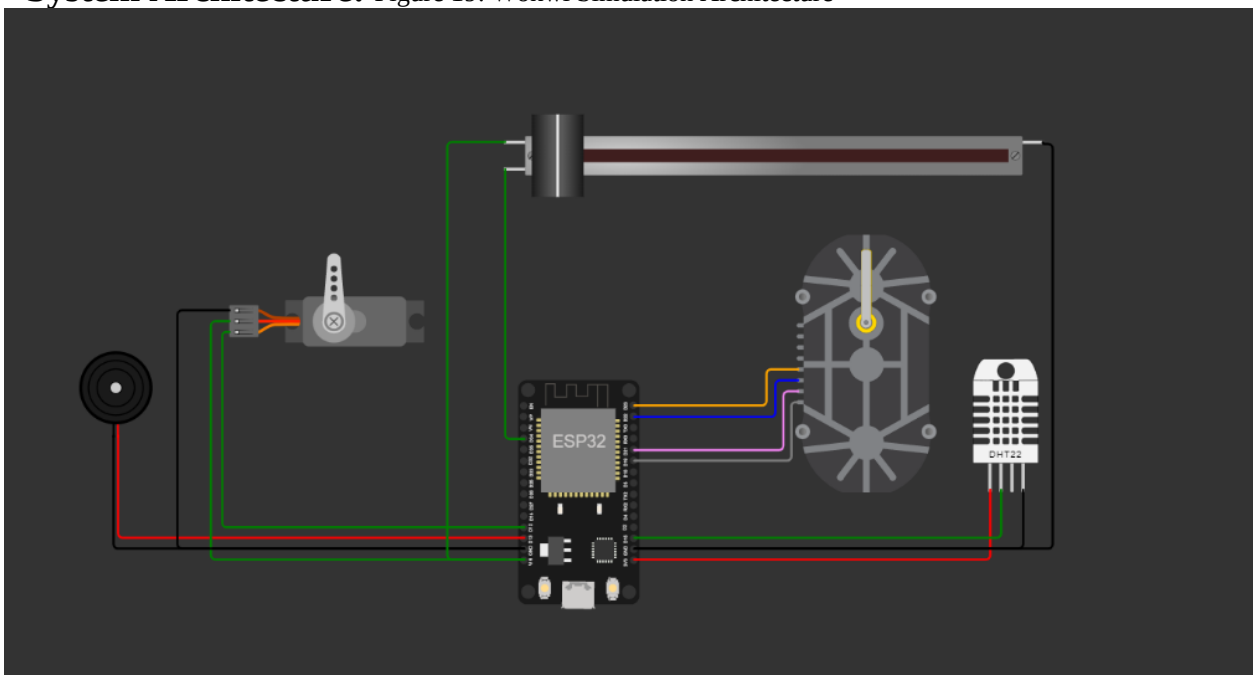


Node RED Dashboard Alert

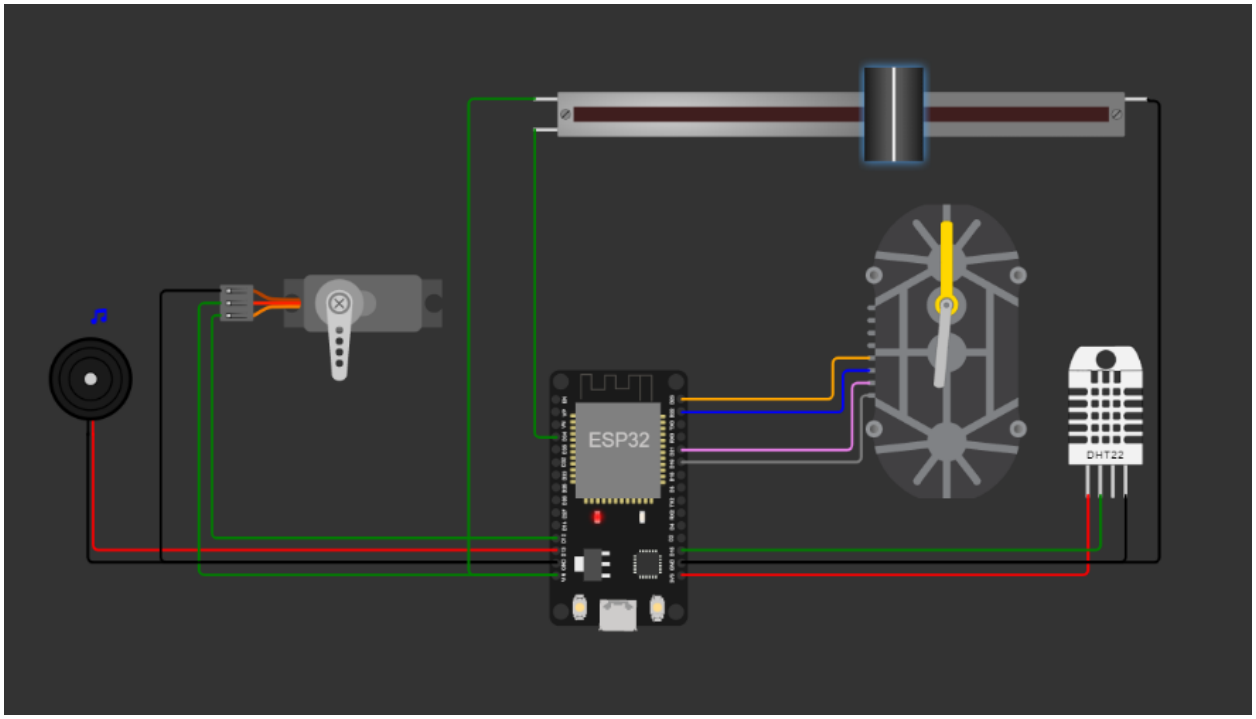
7.2 WOKWI Simulation

The IoT Device is created as a simulation using the WOKWI Platform. The WOKWI platform contains the ESP32 to create a communication connection to the Node-RED. It uses DHT11 sensor to create a temperature and humidity value. And a Gas Sensor (aliased using slide potentiometer) to send the gas level. It displayed the statuses in the Serial Monitor.

System Architecture: Figure 19: Wokwi Simulation Architecture



Gas Leakage Monitoring System (WOKWI Simulation)



Working - Gas Leakage Monitoring System (WOKWI Simulation)

Figure 20: Wokwi Simulation Architecture working

Serial Monitor Output - Gas Leakage Monitoring System (WOKWI Simulation)

Figure 21: Wokwi Simulation Output

Code:

sketch.ino

```
// Team ID - PNT2022TMID19226

// Including Required Header Files
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>
#include <Stepper.h>
#include <ESP32Servo.h>

/*
 NOTE:
 As Gas Sensor is not available in Wokwi platform.
 Slide Potentiometer is used instead of Gas Sensor, to variably set level of gas leakage.
*/

// Defining Constants
#define DHTPIN 15
#define GAS_LEVER 34  // Slide Potentiometer
#define buzzer 13
#define LED 5
const int servoPin = 12;
Servo valve;
DHTesp dhtsensor;
Stepper stepper(1000, 19,21,22,23);


void callback(char* subscribetopic, byte* payload, unsigned int payloadLength);

#define ORG "sgoqkq"
#define DEVICE_TYPE "Gas_Leakage_Detection_Device"
#define DEVICE_ID "Gas_Leakage_Detection_Device1"
#define TOKEN "123456789"
```

```
String data3;
float h, t, g;
int pos=0;
boolean valve_open=true;
//-------- Customise the above values --------
char server[] = ORG ".messaging.internetofthings.ibmcloud.com";
char publishTopic[] = "iot-2/evt/Data/fmt/json";
char subscribetopic[] = "iot-2/cmd/test/fmt/String";
char authMethod[] = "use-token-auth";
char token[] = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;


//-----------------------------------------
WiFiClient wifiClient;
PubSubClient client(server, 1883, callback ,wifiClient);
void setup()
{
  Serial.begin(115200);
  dhtsensor.setup(DHTPIN,DHTesp::DHT22);
  stepper.setSpeed(100);
  valve.attach(servoPin);
  pinMode(GAS_LEVER, INPUT);
  pinMode(buzzer,OUTPUT);
  delay(10);
  Serial.println();
```

```
  wificonnect();
  mqttconnect();
  valve.write(90);

}

void loop()
{
  TempAndHumidity data=dhtsensor.getTempAndHumidity();
  t=data.temperature;
  h=data.humidity;
  g=map(int(analogRead(GAS_LEVER)), 0, 4095, 200, 2000);
  Serial.print("temperature:");
  Serial.println(t);
  Serial.print("Humidity:");
  Serial.println(h);
  Serial.print("Gas Level:");
  Serial.println(g);

  if(g>500){
    tone(buzzer, 1000);
    stepper.step(1000);
    valve.write(180);
  }
  else{
    valve.write(90);
    noTone(buzzer);
  }
  PublishData(t, h, g);
  delay(1000);
```

```
  if (!client.loop()) {
    mqttconnect();
  }
}



/*.....................................retrieving to Cloud.............................*/

void PublishData(float temp, float humid, float gas_level) {
  mqttconnect();
  String payload = "{\"temperature\":";
  payload += temp;
  payload += "," "\"humidity\":";
  payload += humid;
  payload += "," "\"gas_level\":";
  payload += gas_level;
  payload += "}";


  Serial.print("Sending payload: ");
  Serial.println(payload);


  if (client.publish(publishTopic, (char*) payload.c_str())) {
    Serial.println("Publish ok");
  } else {
    Serial.println("Publish failed");
  }
```

```
}
void mqttconnect() {
  if (!client.connected()) {
    Serial.print("Reconnecting client to ");
    Serial.println(server);
    while (!!!client.connect(clientId, authMethod, token)) {
      Serial.print(".");
      delay(500);
    }

     initManagedDevice();
     Serial.println();
  }
}
void wificonnect()
{
  Serial.println();
  Serial.print("Connecting to ");

  WiFi.begin("Wokwi-GUEST", "", 6);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
void initManagedDevice() {
  if (client.subscribe(subscribetopic)) {
    Serial.println((subscribetopic));
    Serial.println("subscribe to cmd OK");
  } else {
    Serial.println("subscribe to cmd FAILED");
  }
}

void callback(char* subscribetopic, byte* payload, unsigned int payloadLength)
{

  Serial.print("callback invoked for topic: ");
  Serial.println(subscribetopic);
  for (int i = 0; i < payloadLength; i++) {
    data3 += (char)payload[i];
  }

  Serial.println("data: "+ data3);

  data3="";


}
```

libraries.txt

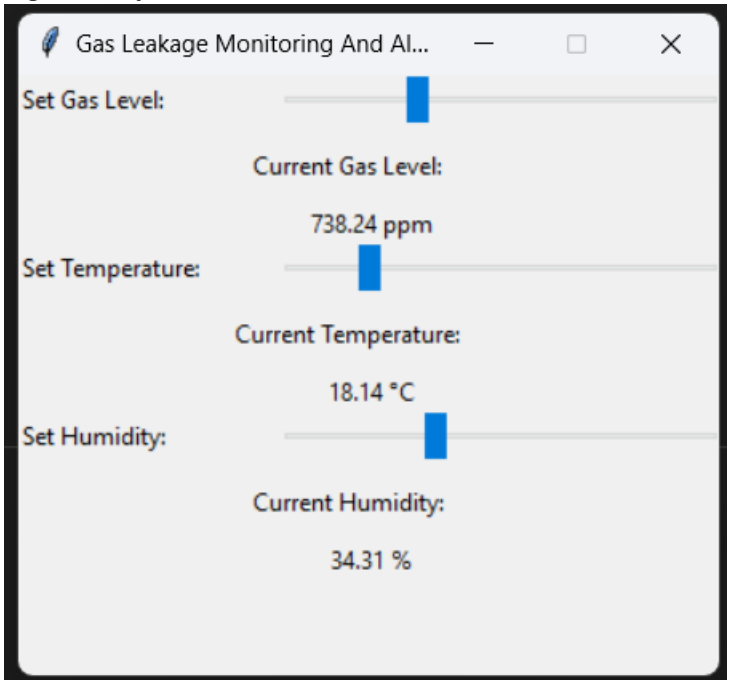
```
# Wokwi Library List
# See https://docs.wokwi.com/guides/libraries

# Automatically added based on includes:
DHT sensor library

PubSubClient
DHT sensor library for ESPx
Servo
Stepper
ESP32Servo
```

7.3 Python Simulation

Python Programming is used to create a simulation of the Gas Leakage and Monitoring System. A GUI is created using the Tkinter Library. Three sliders are created to send the simulated value of the Temperature, Humidity and Gas Level in the atmosphere.

GUI Application:

Figure 22: Python Automation GUI



Code:

```
# Importing Required modules

import time
import sys
import ibmiotf.application  # IBM IoT Watson Platform Module
import ibmiotf.device
import tkinter as tk  # Python GUI Package
from tkinter import ttk  # Python GUI
import time
from threading import Thread

organization = 'sgoqkq'  # Organization ID
deviceType = 'Gas_Leakage_Detection_Device'  # Device type
deviceId = 'Gas_Leakage_Detection_Device1'  # Device ID
authMethod = 'token'  # Authentication Method
authToken = '123456789'  # Replace the authtoken

# Tkinter root window

root = tk.Tk()
root.geometry('350x300')  # Set size of root window
root.resizable(False, False)  # root window non-resizable
root.title('Gas Leakage Monitoring And Alerting System for Industries (PNT2022TMID19226)'
           )

# Layout Configurations

root.columnconfigure(0, weight=1)
root.columnconfigure(1, weight=3)
```

```
# Temperature and Humidity sliders initialization

current_gas = tk.DoubleVar()
current_temp = tk.DoubleVar()
current_humid = tk.DoubleVar()


# slider - temperature and humidity functions

def get_current_gas():  # function returns current gas level value
    return '{: .2f}'.format(current_gas.get())


def get_current_temp():  # function returns current temperature value
    return '{: .2f}'.format(current_temp.get())


def get_current_humid():  # function returns current humidity value
    return '{: .2f}'.format(current_humid.get())


def slider_changed(event):  # Event Handler for changes in sliders
    print '--------'
    print 'Gas Level: {: .2f} , Temperature: {: .2f} , Humidity: {: .2f}'.format(current_gas
.get(),
          current_temp.get(), current_humid.get())
    print '--------'
    gas_label.configure(text=str(get_current_gas()) + ' ppm')  # Displays current gas level as
label content
    temp_label.configure(text=str(get_current_temp()) + ' \xc2\xb0C')  # Displays current
temperature as label content
    humid_label.configure(text=str(get_current_humid()) + ' %')  # Displays current humidity as
label content
```

```
# Tkinter Labels

# label for the gas level slider

slider_gas_label = ttk.Label(root, text='Set Gas Level:')
slider_gas_label.grid(column=0, row=0, sticky='w')

#  Gas Level slider

slider_gas = ttk.Scale(
    root,
    from_=200,
    to=2000,
    orient='horizontal',
    command=slider_changed,
    variable=current_gas,
    )
slider_gas.grid(column=1, row=0, sticky='we')

# current gas level label

current_gas_label = ttk.Label(root, text='Current Gas Level:')
current_gas_label.grid(row=1, columnspan=2, sticky='n', ipadx=10,
                       ipady=10)

# Gas level label (value gets displayed here)

gas_label = ttk.Label(root, text=str(get_current_gas()) + ' ppm')
gas_label.grid(row=2, columnspan=2, sticky='n')


# label for the temperature slider

slider_temp_label = ttk.Label(root, text='Set Temperature:')
slider_temp_label.grid(column=0, row=12, sticky='w')

#  temperature slider

slider_temp = ttk.Scale(
    root,
    from_=0,
    to=100,
    orient='horizontal',
    command=slider_changed,
    variable=current_temp,
    )
slider_temp.grid(column=1, row=12, sticky='we')

# current temperature label

current_temp_label = ttk.Label(root, text='Current Temperature:')
current_temp_label.grid(row=16, columnspan=2, sticky='n', ipadx=10,
                        ipady=10)

# temperature label (value gets displayed here)

temp_label = ttk.Label(root, text=str(get_current_temp()) + ' \xc2\xb0C'
                       )
temp_label.grid(row=17, columnspan=2, sticky='n')
```

```
# label for the humidity slider

slider_humid_label = ttk.Label(root, text='Set Humidity:')
slider_humid_label.grid(column=0, row=20, sticky='w')

#  humidity slider

slider_humid = ttk.Scale(
    root,
    from_=0,
    to=100,
    orient='horizontal',
    command=slider_changed,
    variable=current_humid,
    )
slider_humid.grid(column=1, row=20, sticky='we')

# current humidity label

current_humid_label = ttk.Label(root, text='Current Humidity:')
current_humid_label.grid(row=34, columnspan=2, sticky='n', ipadx=10,
                         ipady=10)

# humidity label (value gets displayed here)

humid_label = ttk.Label(root, text=str(get_current_humid()) + ' %')
humid_label.grid(row=36, columnspan=2, sticky='n')


def publisher_thread():
    thread = Thread(target=publish_data)
    thread.start()


def publish_data():

    # Exception Handling

    try:
        deviceOptions = {
            'org': organization,
            'type': deviceType,
            'id': deviceId,
            'auth-method': authMethod,
            'auth-token': authToken,
            }
        deviceCli = ibmiotf.device.Client(deviceOptions)
    except Exception, e:

    # ..........................................

        print 'Caught exception connecting device: %s' % str(e)
        sys.exit()

    deviceCli.connect()  # Connect to IBM Watson IoT Platform

    while True:
        temp = int(current_temp.get())
        humid = int(current_humid.get())
        gas_level = int(current_gas.get())
```

```
        # Send Temperature & Humidity to IBM Watson IoT Platform

        data = {'gas_level': gas_level, 'temperature': temp,
                'humidity': humid}

        def myOnPublishCallback():
            print ('Published Gas Level = %s ppm' % gas_level,
                   'Temperature = %s C' % temp, 'Humidity = %s %%'
                   % humid, 'to IBM Watson')

        success = deviceCli.publishEvent('event', 'json', data, qos=0,
                on_publish=myOnPublishCallback)
        if not success:
            print 'Not connected to IoTF'
        time.sleep(1)

publisher_thread()

root.mainloop()  # startup Tkinter GUI

# Disconnect the device and application from the cloud

deviceCli.disconnect()
```

7.2 Mobile Application

Mobile Application is created using the MIT App Inventor. Mobile Application is created to display the temperature level, humidity level and gas level in the atmosphere.

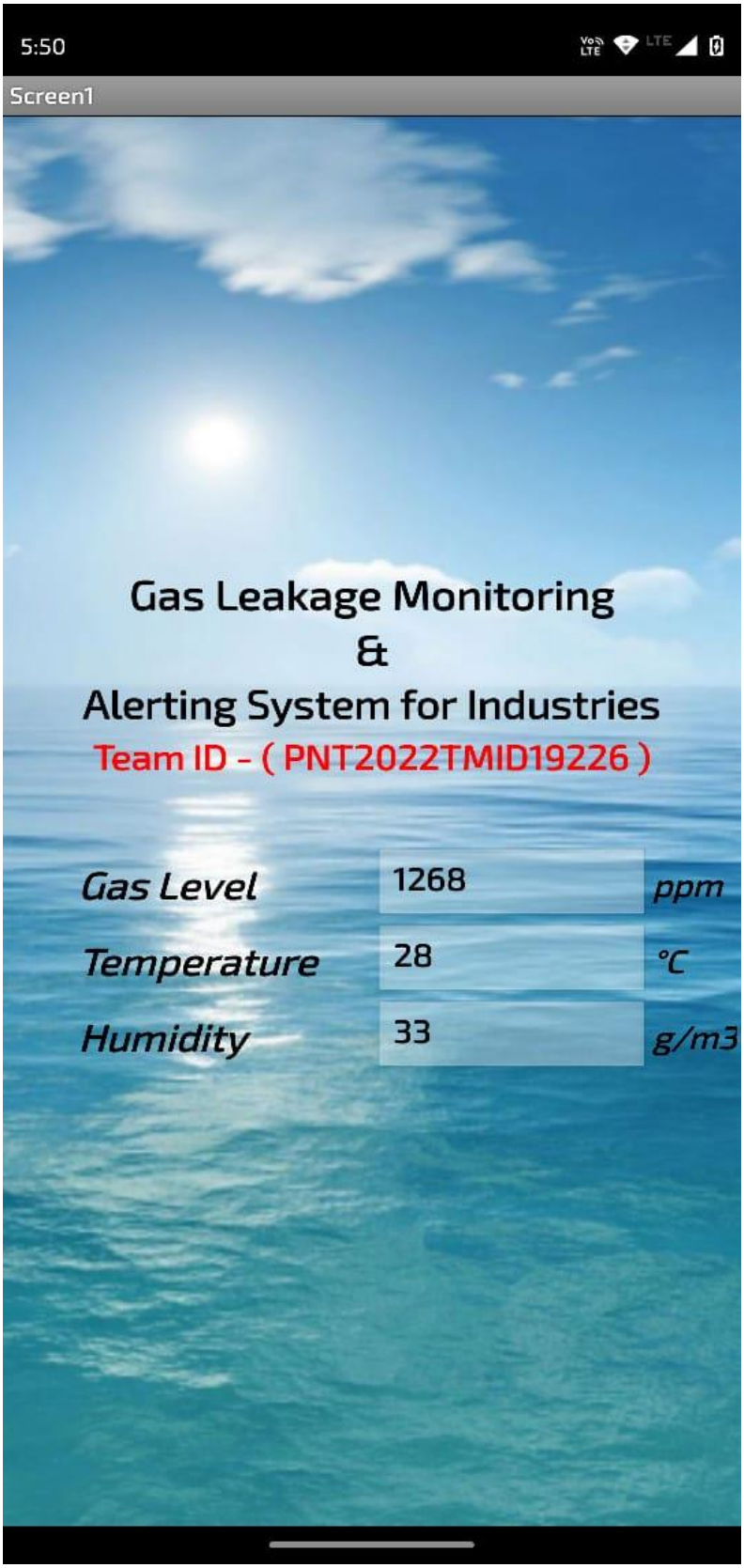


Figure 23: Mobile Application GUI

8. TESTING

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	TC for Automation(Y/N)	BUG ID	Executed By
LoginPage_TC_OO1	UI	SignUp Page	Verify the UI elements in SignUp Page	The Application is hosted.	1.Enter URL and click go 2.Click on SignUp Option. 3.Verify SignUp UI elements: a.username text box b.password text box c.Mobile Number text box d.SignUp button	https://monitor-gas-leakage.web.app/	Application should show below UI elements: a.username text box b.password text box c.Mobile Number text box d.SignUp button	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO2	UI	SignUp Page	Verify the UI elements in SignUp Page for Google Account Login	Google Login must be implemened.	1.Enter URL and click go 2.Click on SignUp Option. 3.Verify the Google Login UI element. a.Google Login Button	https://monitor-gas-leakage.web.app/	Application should show below UI elements: a. Google Login Button	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO3	UI	SignUp Page	Verify the UI elements in SignUp Page for Facebook Account Login	Facebook Login must be implemented.	1.Enter URL and click go 2.Click on SignUp Option. 3.Verify the Facebook Login UI element. a.Facebook Login Button	https://monitor-gas-leakage.web.app/	Application should show below UI elements: a. Facebook Login Button	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO4	Functional	SignUp page	Verify user is able to signup into application with Valid credentials	The user must have a valid mail id and password.	1.Enter URL and click go 2.Click on SignUp option. 3.Enter email in Email text box 4.Enter password in password text box 5.Click on signup button	Username: hari@gmail.com password: Testing123	User should navigate to verify email page.	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO5	UI	SignUp Page	Verify whether the user is prompted with "Verify Email" message.	The user must have a valid mail id and password.	1.Enter URL and click go 2.Click on SignUp option. 3.Enter email in Email text box 4.Enter password in password text box 5.Click on signup button	Username: hari@gmail.com password: Testing123	Verify whether the user is prompted with "Verify Email" message. Application should show below UI elements: a. "Verify Email" label box b. "Refresh" Button c. "Resend Verification Email" button	Working as expected	Pass		Y		Hariboobaalan P N
LoginPage_TC_OO6	Functional	SignUp Page	Verify whether the user receives verification mail.	The user must have a valid mail id and password.	1.Open the Mail account. 2.Check for account verification mail from gas-leakage-monitoring app. 3.Click on verification link for mail verification.	Verification link	Verify whether the user is able to verify the email id using the link and redirect to gas-leakage-monitoring application.	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO7	Functional	SignUp Page	Verify whether the Refresh button refreshes the page		1. Click on the Refresh Button.		Verify whether the page refreshes	Working as expected	Pass		Y		Hariboobaalan P N
LoginPage_TC_OO8	Functional	SignUp Page	Verify whether the Resend Verification Email button resends the verification email to the user mail id.	The user must have a valid mail id and password.	1.Click on the Resend Verification Mail button.		Verify whether the user receives a verification mail from gas-leakage-monitoring application.	Working as expected	Pass		Y		Hariboobaalan P N
LoginPage_TC_OO9	Functional	SignUp Page	Verify whether the user is able to see the dashboard after email verification.	The user must have a valid mail id and password.	1. After mail ID verification redirect to web application. 2. Click on refresh button.		Verify whether the user is able to see the dashboard.	Working as expected	Pass		N		Hariboobaalan P N
LoginPage_TC_OO10	UI	Login Page	Verify the UI elements in Login Page	UI Components must be embedded.	1.Enter URL and click go 2.Verify Login Page UI elements: a.username text box b.password text box c.Log In button		Application should show below UI elements: a.username text box b.password text box c.Log In button	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO11	Functional	Login Page	Verify user is able to login into application with Valid credentials	The user must have a valid mail id and password.	1.Enter URL and click go 2.Enter email in Email text box 3.Enter password in password text box 4.Click on log In button	Username: hari@gmail.com password: Testing123	1.Application should display Login Successful. 2.User should be redirected to dashboard page.	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO12	Functional	Login Page	Verify user is able to login into application with InValid credentials	Google Auth must be implemented.	1.Enter URL and click go 2.Enter invalid email in Email text box 3.Enter password in password text box 4.Click on log In button	Username: gef@gmail.com password: Testing123	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO13	Functional	Login Page	Verify user is able to login into application with InValid credentials	The App is hosted.	1.Enter URL and click go 2.Enter email in Email text box 3.Enter invalid password in password text box 4.Click on log In button	Username: hari@gmail.com password: abced	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO14	UI	Login Page	Verify the UI elements in Login Page for Google Account Login	UI Components must be embedded.	1.Enter URL and click go 2.Verify the Google Login UI element. a.Google Login Button	https://monitor-gas-leakage.web.app/	Application should show below UI elements: a. Google Login Button	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO15	Functional	Login Page	1. Verify whether the google login button redirects the user to Google Auth page. 2. Verify whether the user is able to enter the gmail id and password.	The user must have a valid Gmail id and password.	1. Enter the URL and click go. 2. Click Google Login Button. 3. Enter the Gmail ID and Password in the pop up window. 4. Click Enter.	Username: hari@gmail.com password: Testing123	1. Application should login using the user's Google Account credentials. 2. User should be able to view the dashboard page.	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO16	Functional	Login Page	Verify whether the user is able to login using invalid credentials.	Google Auth must be implemented.	1. Enter the URL and click go. 2. Click Google Login Button. 3. Enter the invalid Gmail ID. 4. Enter the password. 5. Click login.	Username: gef@gmail.com password: Testing123	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO17	Functional	Login Page	Verify whether the user is able to login using invalid credentials.	Google Auth must be implemented.	1. Enter the URL and click go. 2. Click Google Login Button. 3. Enter the Gmail ID. 4. Enter the invalid password. 5. Click login.	Username: hari@gmail.com password: abced	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO18	UI	Login Page	Verify the UI elements in SignUp Page for Facebook Account Login	The user must have a valid Facebook id and password.	1.Enter URL and click go 2.Verify the Facebook Login UI element. a.Facebook Login Button	https://monitor-gas-leakage.web.app/	Application should show below UI elements: a. Facebook Login Button	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO19	Functional	Login Page	1. Verify whether the facebook login button redirects the user to facebook Auth page. 2. Verify whether the user is able to enter the facebook id and password.	The user must have a valid Facebook id and password.	1. Enter the URL and click go. 2. Click facebook Login Button. 3. Enter the facebook ID and Password in the pop up window. 4. Click Enter.	Username: hari1234 password: Testing123	1. Application should login using the user's facebook Account credentials. 2. User should be able to view the dashboard page.	Working as expected	Pass		N		Karthikeyan
LoginPage_TC_OO20	Functional	Login Page	Verify whether the user is able to login using invalid credentials.	Facebook Auth must be implemented.	1. Enter the URL and click go. 2. Click facebook Login Button. 3. Enter the Invalid facebook ID. 4. Enter the password. 5. Click login.	Username: gef4231 password: Testing123	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO21	Functional	Login Page	Verify whether the user is able to login using invalid credentials.	Facebook Auth must be implemented.	1. Enter the URL and click go. 2. Click facebook Login Button. 3. Enter the facebook ID. 4. Enter the Invalid password. 5. Click login.	Username: hari1234 password: abced	1.Application should display Invalid Username/Password message.	Working as expected	Pass		Y		Karthikeyan
LoginPage_TC_OO22	Functional	Database	Verify the user credentials are stored in the realtime database.	Realtime Database must be hosted. And the application must be connected to database.			The valid user credentials must be stored in the realtime database for future use.	Working as expected	Pass		Y		Harithaa G
LoginPage_TC_OO23	UI	Dashboard	Verify the components of the dashboard.	UI Components must be embedded.	1.Enter the URL 2.Login to the application. 3.View the components of the dasboard. a.Team ID (PNT2022TMID19226) label b.Temperature Sensor Gauge Status c.Humidity Sensor Gauge Status d.Gas Level Sensor Gauge Status e.Service Request Option		Application should show below UI elements: a.Team ID (PNT2022TMID19226) label b.Temperature Sensor Gauge Status c.Humidity Sensor Gauge Status d.Gas Level Sensor Gauge Status e.Service Request Option f.SignOut button	Working as expected	Pass		N		Harithaa G
LoginPage_TC_OO24	Functional	Dashboard	Verify the working of Gas Level Sensor	NodeRED must be configured.	1. Enter URL 2. Login to Application. 3. Goto Dashboard. 4. View the status of the sensor.		The gauge must graphically display the correct accurate reading of the gas level sensor.	Working as expected	Pass		Y		Harithaa G
LoginPage_TC_OO25	Functional	Dashboard	Verify the working of Humidity Sensor	NodeRED must be configured.	1. Enter URL 2. Login to Application. 3. Goto Dashboard. 4. View the status of the sensor.		The gauge must graphically display the correct accurate reading of the humidity level sensor.	Working as expected	Pass		Y		Harithaa G

LoginPage_TC_OO26	Functional	Dashboard	Verify the working of Temperature Sensor	NodeRED must be configured.	1. Enter URL 2. Login to Application. 3. Goto Dashboard. 4. View the status of the sensor.		The gauge must graphically display the correct accurate reading of the temperature level sensor.	Working as expected	Pass		Y		Harithaa G
LoginPage_TC_OO27	Functional UI	Dashboard	Verify whether the user is prompted with alert if gas leak is detected	NodeRED must be configured.	1. Enter URL. 2. Login to Application. 3. Goto Dashboard. 4. View the status of the sensor.		The application must alert the user with a prompt message "Gas Leakage Detected" in case of any gas leakage.	Working as expected	Pass		Y		Harithaa G
LoginPage_TC_OO28	Functional	SMS	Verify whether the user receives alert message incase of gas leakage	SMS Notification Alert System must be implemented.			The user receives a SMS notification in case of any gas leakage.	Working as expected	Pass		Y		Harithaa G
LoginPage_TC_OO29	UI	Service Request	Verify whether the user is displayed with options for service when service request button is clicked.	Service Request option must be implemented.	1. Enter URL. 2. Login to Application. 3. Goto Dashboard. 4. Click on service button		The user is displayes with a popup for service request. The user is displayed with UI components for a. Service Request Button b. ScalingUp Request Button c. Back to Dashboard Button	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO30	Functional	Service Request	Verify the working of Service request.	Service Request option must be implemented.	1. Enter URL 2. Login to Application. 3. Goto Dashboard. 4. Click on service button 5. Click on Service Request Button.		1.The request for service is sent to the service provider. 2.The user is prompted with "Request Sent" message.	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO31	Functional	Service Request	Verify the working of ScalingUP request.	Service Request option must be implemented.	1. Enter URL. 2. Login to Application. 3. Goto Dashboard. 4. Click on service button 5. Click on ScalingUp Request Button.		1.The request for scalingUp is sent to the service provider. 2.The user is prompted with "Request Sent" message.	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO32	Functional	Service Request	Verify the working of Back to Dashboard Button.	Service Request option must be implemented.	1. Enter URL. 2. Login to Application. 3. Goto Dashboard. 4. Click on service button 5. Click on Back to Dashboard Button.		The user is redirected to the dashboard.	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO33	Functional	Sign Out	Verify the working of signOut button.	Account Signout Feature must be implemented.	1. Enter URL. 2. Login to Application. 3. Goto Dashboard. 4. Click on signout button.		The user is signed out from the application.	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO34	UI	Mobile Application	Verify the UI Components of the Mobile Application	Mobile Application must be developed and installed.Mobile application must be connected with NodeRED using HTTP Requests.	1. Download and install the mobile Application. 2. Open the mobile Application.	Mobile Apk	The user is displayed with the Mobile Application UI components. 1. Team ID (PNT2022TMID19226) label. 2. Temperature Level Label. 3. Gas Level Label. 4. Humidity level Label. 5. Textbox to display temperature level in degree celsius. 6. Textbox to display gas level in ppm. 7. Textbox to display humidity level in units.	Working as expected	Pass		N		Kavin P
LoginPage_TC_OO35	Functional	Mobile Application	Verify whether the mobile application shows the realtime monitoring of the status of the gas leakage monitoring system	Mobile Application must be developed and installed.Mobile application must be connected with NodeRED using HTTP Requests.	1. Download and install the mobile Application. 2. Open the mobile Application.	Mobile Apk	The mobile application displays the realtime status of the sensors in the gas leakage monitoring system.	Working as expected	Pass		Y		Kavin P
LoginPage_TC_OO36	Functional	SMS	Verify whether the user receives SMS notification incase of gas leakage detection.	SMS Notification Alert System must be implemented.			The user receives a SMS notification in case of any gas leakage.	Working as expected	Pass		Y		Kavin P

8.2UAT

Defect Analysis

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	8	4	1	0	13
Duplicate	1	0	3	0	4
External	4	3	0	1	8
Fixed	13	7	4	1	25
Not Reproduced	0	0	1	0	1
Skipped	1	0	0	0	1
Won't Fix	0	0	0	0	0
Totals	27	14	9	2	52

Test Case Analysis

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	3	0	0	3
Client Application	36	0	0	36
Security	4	0	0	4
Outsource Shipping	1	0	0	1
Exception Reporting	10	0	0	10
Final Report Output	4	0	0	4
Version Control	1	0	0	1

9. RESULTS

9.1 Performance Metrics

CPU Usage:

The micro version of python makes the most efficient use of the CPU. The program runs in O (1) timefor each loop, ignoring the network and communication. To improve communication with MQTT, the programsleeps every 1 second. Because the program runs in O (1) time and the compiler optimizes it during compilation, there is less CPU load per cycle. The following instructions are stored on the stack memory and can be popped after execution.

Memory Usage:

The sensor values and networking data are saved in the ESP32's sram. It's a lot of information because the ESP32 only has 520 KB of memory. To save memory and ensure optimal program execution, the exact addresses for each memory cycle are overwritten with new values.

Error Rates:

The error rates are very low because the backend and dashboard are handled with node-red. Exceptions are handled properly so that the system's usability is not affected.

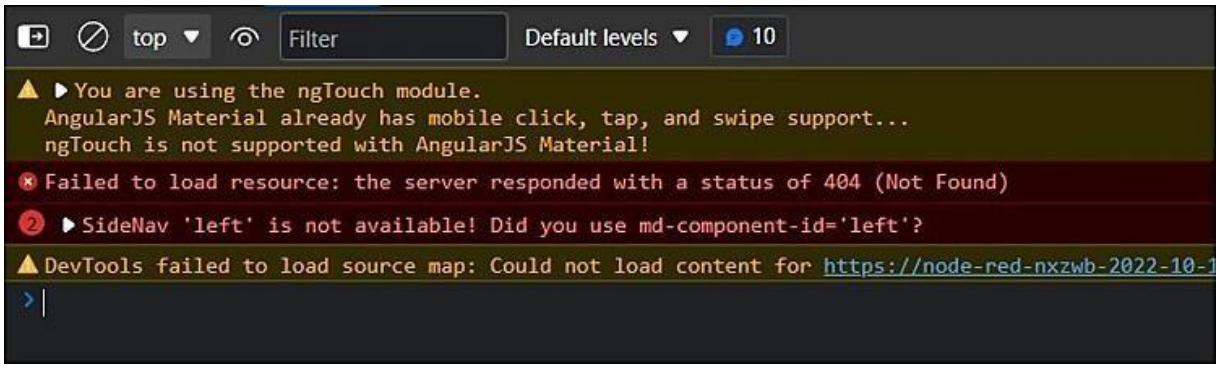


Figure 24: Console Output

Latency and Respose Time:

The DOM handling of the received data is optimal and latency is low .After the DOM is loaded the entire site is loaded to the browser.

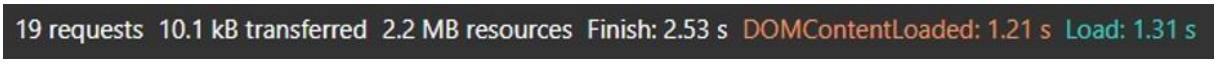


Figure 25: Latency & Response Time Output

In addition, the server responds quickly. The average response time is acceptable.

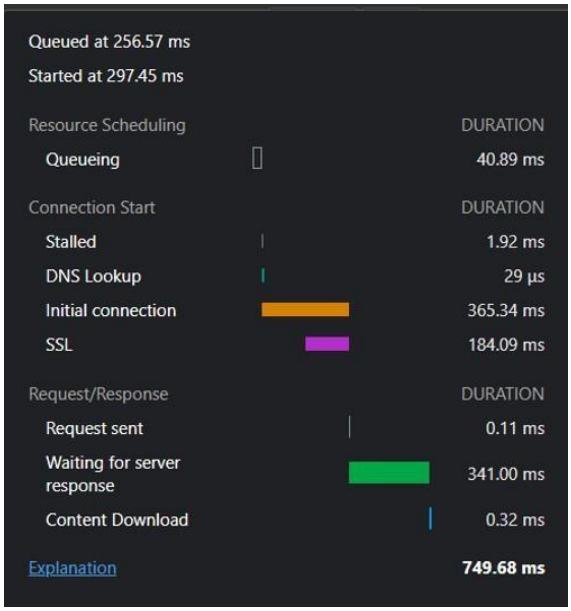


Figure 26: Response Time Output

For the data sent from the IoT Device (considering the sleep of one second from the IoT), the response is much faster. We can see the delay caused by the sleep function.

The average time is well over optimal value

Average Time = (5ms + 2600ms)/2

= 1302.5

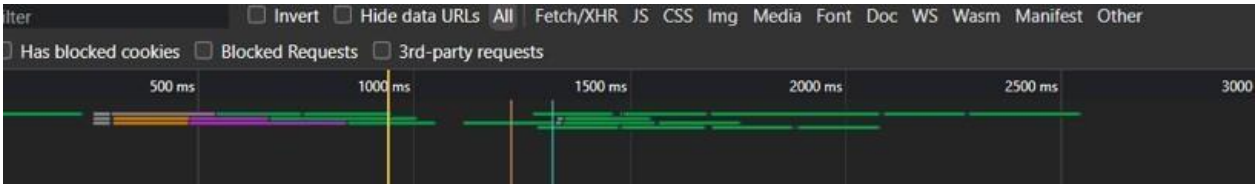


Figure 27: Average Time Output

Garbage Collection:

The Node framework handles garbage collection on the server side. C++ does not have garbage collection features in IoT devices. However, in this case, it is not necessary because the memory will be used again to store the data. There is no allocation of any dangling pointers or poorly handled address space.

10. ADVANTAGES & DISADVANTAGES

Advantages:

- Get Real-time alerts about the gaseous level presence in the atmosphere.
- Get Real-time status of gas sensor, temperature sensor & humidity sensor.
- Prevent fire hazards.
- Real-time alert and notification system in case of any gas leakages.
- Cost-effective installation.
- Get immediate SMS notification incase of any gas leakage.
- Immediate Response for Service Request and Scaling Up Request.
- Users can access the dashboard through a web application.

Disadvantages:

- Internet access is always necessary just to deliver the SMS alert
- The entire operation fails if the hardware device malfunctions.
- Most elemental organic vapors are toxic to the sensor.

11. CONCLUSION

By developing a smart gas leakage monitoring and alerting system that actively monitors for gas leaks, & alerts the workers of the industry in case of detection of any gas leakage & also sends SMS notifications to administrators and also takes precautionary measures like turning ON exhaust fans so, we can therefore conclude that our problem premise is solved utilizing IoT devices.

12. FUTURE SCOPE

While leakage of inflammable gases can result in fire accidents and cause major loss of human life and damage to both life and property in both households and huge industries, the Gas Leakage Monitoring and Alerting System for Industries can be used to solve this issue. The existing devices can be scaled up for use in houses and large labs, as well as in public spaces and automobiles. In Future sprinkler automation and alert to nearby fire station can be implemented.

13. APPENDIX

ESP32 - Microcontroller:

The ESP32 is a low-cost, low-power system-on-a-chip microcontroller family with integrated Wi-Fi and dual-mode Bluetooth.

- Memory: 320 KiB SRAMCPU: Tensilica Xtensa LX6 Microprocessor @ 160 or 240 MHz
- Power: 3.3 VDC
- Manufacturer: Espressif Systems
- Predecessor: ESP8266

Sensors:

DHT22 - Temperature & Humidity Sensor:

The DHT22 is a simple and inexpensive digital temperature and humidity sensor. It measures the surrounding air with a capacitive humidity sensor and a thermistor and outputs a digital signal on the data pin (no analog input pins needed).

MQ5 - Gas Sensor:

Gas sensors (also referred to as gas detectors) are electronic devices that detect and identify various types of gasses. They are frequently used to detect toxic or explosive gases as well as to measure gas concentration.

Github Link:

IBM-EPBL/IBM-Project-5462-1658765976: Gas Leakage monitoring & Alerting system for Industries (github.com)

Demo Video:

https://github.com/IBM-EPBL/IBM-Project-5462-1658765976/blob/main/Final%20Deliverables/PNT2022TMID19226_Final-Demo.mp4?raw=true