EE331

EMBEDDED SYSTEMS LAB

Final Project Report

IoT based alert and mail sending system

Group Number 2

Utsav Bansal (2104236)

Manvendra Singh (2104216)

Sujal Jani (2104212)

Project Guide:

1)Dr.Sunil Dutt, assistance Professor

2) Parmeshwar, Technical Superintendent

3) Diptesh R Naik, Technical Assistant

4) Pratik G and Tanu Kumari, TA

INDEX

INTRODUCTION

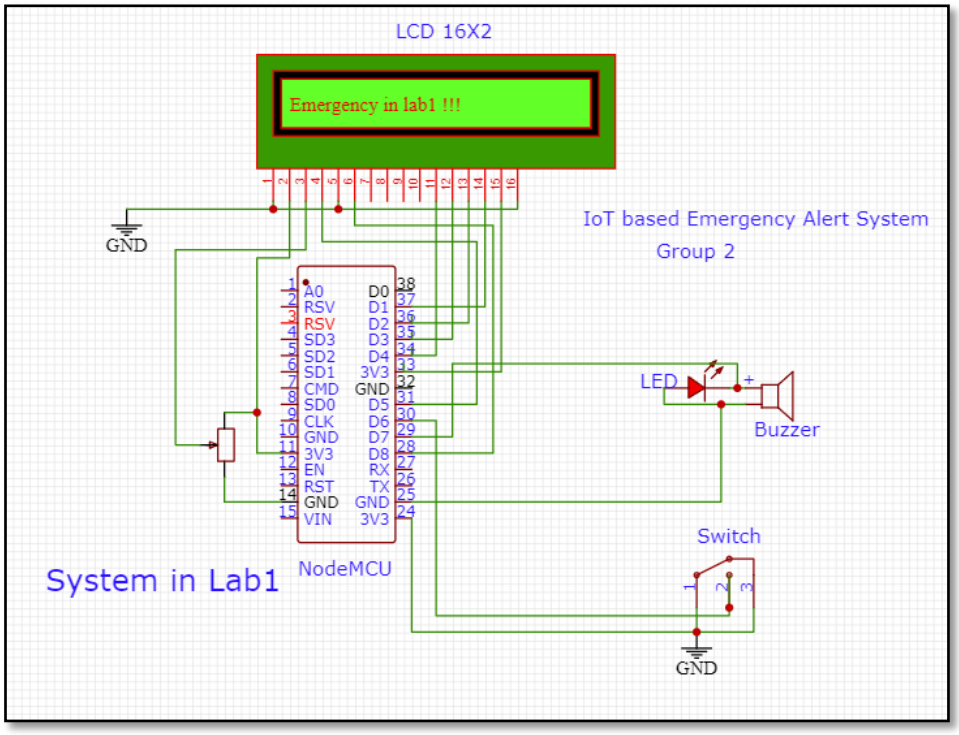
The integration of IoT technology with a physical alert mechanism represents a significant advancement in emergency response systems. Our lab report explores the development and implementation of such a system with the primary objective of providing quick and effective means of alerting personnel or relevant authorities during critical situations. With the capability to automatically trigger alerts, whether it be in response to a fire outbreak, medical emergency, or any other urgent scenario, this system ensures a prompt and coordinated response. By harnessing the power of Internet of Things (IoT) devices, such as the NodeMCU, in conjunction with traditional hardware components like switches and buzzers, our system bridges the gap between the digital and physical realms, enhancing overall responsiveness and reliability. Additionally, the system's ability to seamlessly send emails to the security office further enhances its utility, enabling them to provide timely assistance and support when needed most. In this report, we delve into the programming aspects and technical details of our system, highlighting its potential to mitigate risks and minimize damage during emergencies.

COMPONENTS

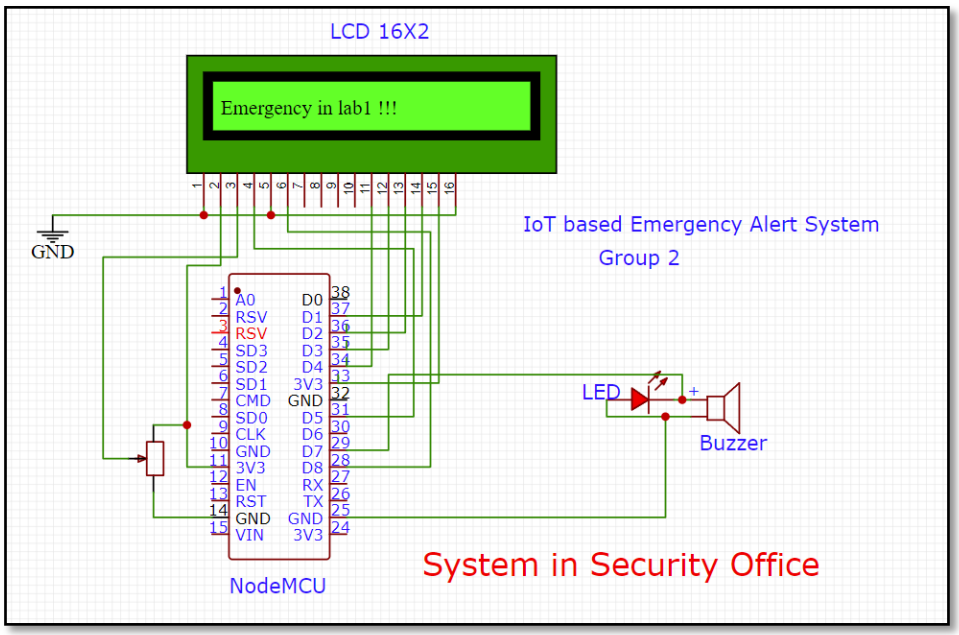
S. No.	Component Name	Quantity	Explanation
1	NodeMCU ESP6266	2	The central control unit responsible for interfacing with the internet and managing the alert system.
2	ON-OFF Button	2	A physical switch used to manually activate the emergency alert system when needed
3	LCD Display	2	Displays the status of the message/email sent to the relevant authority.
4	LEDs	2	LED will turn ON on pressing the switch button
5	Buzzer	2	An audible alarm that emits a loud sound to grab attention and signal the occurrence of an emergency.
6	BreadBoard	2	-
7	Jumper Wires	-	-

CIRCUIT DIAGRAM

- Circuit diagram of Lab



- Circuit diagram of Security Office

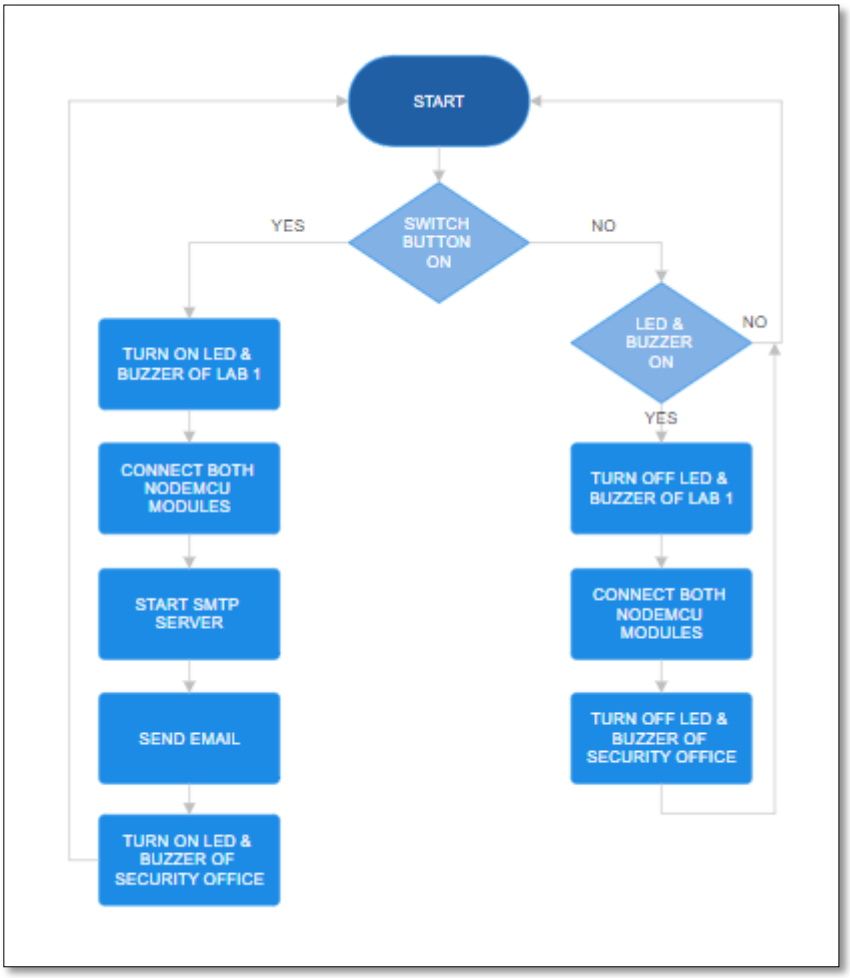


- Circuit Diagram explanation:

  In the above circuit diagrams, the LED is connected in parallel with the buzzer and is linked through the digital pin of the NodeMcu. A three-pin switch is connected between them to enable or disable the circuit flow at any time. An LCD is connected to display the status on the screen. When the switch is ON, the buzzer will ring and the LED will glow, signifying the state of emergency and the start of the emergency protocol.

  The circuit diagram of the security room is similar to the above. The buzzer will be ON when a message is sent. The LCD in the security room will display the emergency message from Lab 1.

FLOWCHART



Explanation- During a state of emergency, the switch will be turned ON to initiate the emergency protocol. The sound of the buzzer will grab the attention of the surroundings, and the glow of the LED will signify the start of sending a message. During this interval, the NodeMcu modules on both sides will be connected with Wi-Fi. The SMTP server will then send an email to the security room's mail. When the mail is received, the buzzer on their side will be turned ON, notifying them with the emergency message on the LCD.

After that, when the switch is turned OFF, the buzzer and LED of Lab 1 will be turned off. Along with it, the buzzer on the security's side will be turned OFF since the NodeMcu modules are connected together.

CODE

- Lab system code

```
#include <ezButton.h>
#include <Arduino.h>
#if defined(ESP32) || defined(ARDUINO_RASPBERRY_PI_PICO_W)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#elif __has_include(<WiFiNINA.h>)
#include <WiFiNINA.h>
#elif __has_include(<WiFi101.h>)
#include <WiFi101.h>
#elif __has_include(<WiFiS3.h>)
#include <WiFiS3.h>
#endif
#include <LiquidCrystal.h>

WiFiClient TCPclient;

// Creates an LCD object. Parameters: (rs, enable, d4, d5, d6, d7)
LiquidCrystal lcd(D8, D5, D4, D3, D2, D1);

#include <ESP_Mail_Client.h>

#define WIFI_SSID "Utsav-PC"
#define WIFI_PASSWORD "utsavbansal"

const char* ssid = "Utsav-PC";     // CHANGE TO YOUR WIFI SSID
const char* password = "utsavbansal"; // CHANGE TO YOUR WIFI PASSWORD
const char* serverAddress = "192.168.137.55"; // CHANGE TO ESP32#2'S IP ADDRESS
const int serverPort = 4080;
ezButton button(D6); // create ezButton

/** For Gmail, the app password will be used for log in
 *  Check out https://github.com/mobizt/ESP-Mail-Client#gmail-smtp-and-imap-required-app-passwords-to-sign-in
 *
 * For Yahoo mail, log in to your yahoo mail in web browser and generate app password by go to
 * https://login.yahoo.com/account/security/app-passwords/add/confirm?src=noSrc
 *
 * To use Gmai and Yahoo's App Password to sign in, define the AUTHOR_PASSWORD with your App Password
 * and AUTHOR_EMAIL with your account email.
 */

/** The smtp host name e.g. smtp.gmail.com for GMail or smtp.office365.com for Outlook or smtp.mail.yahoo.com
*/
#define SMTP_HOST "smtp.gmail.com"

/** The smtp port e.g.
 * 25  or esp_mail_smtp_port_25
 * 465 or esp_mail_smtp_port_465
 * 587 or esp_mail_smtp_port_587
 */
#define SMTP_PORT 465

/* The log in credentials */
#define AUTHOR_EMAIL "alerts.iitgoa@gmail.com"     // Email of the sender
#define AUTHOR_PASSWORD "ksnpkfsesgcicvlz"         //Password of the sender

/* Recipient email address */
//#define RECIPIENT_EMAIL "fanfukra31@gmail.com"
```

```
//String RECIPIENT_EMAIL[2][2] = {{"Security Office","fanfukra31@gmail.com"},{"Utsav","text@abcd.xyz"}};
String RECIPIENT_EMAIL[2][2] = {{"Security
Office","utsavbansal75@gmail.com"},{"Sujal","jani.sujal.21042@iitgoa.ac.in"}};      // Reciever's email
address

/* Declare the global used SMTPSession object for SMTP transport */
SMTPSession smtp;

/* Callback function to get the Email sending status */
void smtpCallback(SMTP_Status status);

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
WiFiMulti multi;
#endif

void setup()
{
  pinMode(D7,OUTPUT);
  pinMode(D6,INPUT);
  lcd.begin(16, 2);
  lcd.clear();

  Serial.begin(115200);

#if defined(ARDUINO_ARCH_SAMD)
  while (!Serial)
    ;
#endif

  Serial.println();

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
  multi.addAP(WIFI_SSID, WIFI_PASSWORD);
  multi.run();
#else
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
#endif

  Serial.print("Connecting to Wi-Fi");

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
  unsigned long ms = millis();
#endif

  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(300);
#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
    if (millis() - ms > 10000)
      break;
#endif
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  if (TCPclient.connect(serverAddress, serverPort)) {
    Serial.println("Connected to TCP server");
  } else {
    Serial.println("Failed to connect to TCP server");
  }


}
```

```
  /** The html text message character set e.g.
   * us-ascii
   * utf-8
   * utf-7
   * The default value is utf-8
   */

  /** The content transfer encoding e.g.
   * enc_7bit or "7bit" (not encoded)
   * enc_qp or "quoted-printable" (encoded)
   * enc_base64 or "base64" (encoded)
   * enc_binary or "binary" (not encoded)
   * enc_8bit or "8bit" (not encoded)
   * The default value is "7bit"
   */

  /** The message priority
   * esp_mail_smtp_priority_high or 1
   * esp_mail_smtp_priority_normal or 3
   * esp_mail_smtp_priority_low or 5
   * The default value is esp_mail_smtp_priority_low
   */

  /** The Delivery Status Notifications e.g.
   * esp_mail_smtp_notify_never
   * esp_mail_smtp_notify_success
   * esp_mail_smtp_notify_failure
   * esp_mail_smtp_notify_delay
   * The default value is esp_mail_smtp_notify_never
   */
  // message.response.notify = esp_mail_smtp_notify_success | esp_mail_smtp_notify_failure |
esp_mail_smtp_notify_delay;

  /* Set the custom message header */


  /* Set the TCP response read timeout in seconds */
  // smtp.setTCPTimeout(10);

  /* Connect to the server */

int switch_status_old = 0;
void loop()
{

  if (!TCPclient.connected()) {
    Serial.println("Connection is disconnected");
    TCPclient.stop();

    // reconnect to TCP server (Arduino #2)
    if (TCPclient.connect(serverAddress, serverPort)) {
      Serial.println("Reconnected to TCP server");
    } else {
      Serial.println("Failed to reconnect to TCP server");
    }
  }

//Serial.println("value of D6");
//Serial.println(digitalRead(D6));
//Serial.println("Yes");

  if(digitalRead(D6)==1){            // If the switch is closed than this function will run
    TCPclient.write('1');
    TCPclient.flush();
    //Serial.println("- The button is pressed, sent command: 1");

    digitalWrite(D7,HIGH);          // Turns on LED and Buzzer
```

```
    if (switch_status_old == 0){    // Checks if switch is been made high recently or not
      switch_status_old = 1;        // Makes switch_status_old = 1 so that in next loops, untill switch is
turned off and than turned on, email wont be sent
      Serial.println("hello");
      sendEmail();
    }

  }
  else{

    TCPclient.write('0');
    TCPclient.flush();
    //Serial.println("- The button is released, sent command: 0");

    if (switch_status_old == 1){
      switch_status_old = 0;           // So that next time when switch is turned on, email is been send
    }
    digitalWrite(D7,LOW);        // turns off led and buzzer
    lcd.clear();
    }
}

/* Callback function to get the Email sending status */
void smtpCallback(SMTP_Status status)
{
  /* Print the current status */
  Serial.println(status.info());

  /* Print the sending result */
  if (status.success())
  {
    // MailClient.printf used in the examples is for format printing via debug Serial port
    // that works for all supported Arduino platform SDKs e.g. SAMD, ESP32 and ESP8266.
    // In ESP8266 and ESP32, you can use Serial.printf directly.

    Serial.println("----------------");
    MailClient.printf("Message sent success: %d\n", status.completedCount());
    MailClient.printf("Message sent failed: %d\n", status.failedCount());
    Serial.println("----------------\n");

    for (size_t i = 0; i < smtp.sendingResult.size(); i++)
    {
      /* Get the result item */
      SMTP_Result result = smtp.sendingResult.getItem(i);

      // In case, ESP32, ESP8266 and SAMD device, the timestamp get from result.timestamp should be valid if
      // your device time was synched with NTP server.
      // Other devices may show invalid timestamp as the device time was not set i.e. it will show Jan 1,
1970.
      // You can call smtp.setSystemTime(xxx) to set device time manually. Where xxx is timestamp (seconds
since Jan 1, 1970)

      MailClient.printf("Message No: %d\n", i + 1);
      MailClient.printf("Status: %s\n", result.completed ? "success" : "failed");
      MailClient.printf("Date/Time: %s\n", MailClient.Time.getDateTimeString(result.timestamp, "%B %d, %Y
%H:%M:%S").c_str());
      MailClient.printf("Recipient: %s\n", result.recipients.c_str());
      MailClient.printf("Subject: %s\n", result.subject.c_str());
    }
    Serial.println("----------------\n");

    // You need to clear sending result as the memory usage will grow up.
    smtp.sendingResult.clear();
  }
}

void sendEmail() {
```

```
  lcd.print("Email is been");         // Prints on the LCD the status of mail
  lcd.setCursor(0, 1);
  lcd.print("sending");
  /*  Set the network reconnection option */
  MailClient.networkReconnect(true);

  // The WiFi credentials are required for Pico W
  // due to it does not have reconnect feature.
#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
  MailClient.clearAP();
  MailClient.addAP(WIFI_SSID, WIFI_PASSWORD);
#endif

  /** Enable the debug via Serial port
   * 0 for no debugging
   * 1 for basic level debugging
   *
   * Debug port can be changed via ESP_MAIL_DEFAULT_DEBUG_PORT in ESP_Mail_FS.h
   */


  for (int i = 0; i < sizeof(RECIPIENT_EMAIL) / sizeof(RECIPIENT_EMAIL[0]); i++) {
    smtp.debug(1);

    /* Set the callback function to get the sending results */
    smtp.callback(smtpCallback);

    /* Declare the Session_Config for user defined session credentials */
    Session_Config config;

    /* Set the session config */
    config.server.host_name = SMTP_HOST;
    config.server.port = SMTP_PORT;
    config.login.email = AUTHOR_EMAIL;
    config.login.password = AUTHOR_PASSWORD;

    /** Assign your host name or you public IPv4 or IPv6 only
    * as this is the part of EHLO/HELO command to identify the client system
    * to prevent connection rejection.
    * If host name or public IP is not available, ignore this or
    * use loopback address "127.0.0.1".
    *
    * Assign any text to this option may cause the connection rejection.
    */
    config.login.user_domain = F("127.0.0.1");

    /*
    Set the NTP config time
    For times east of the Prime Meridian use 0-12
    For times west of the Prime Meridian add 12 to the offset.
    Ex. American/Denver GMT would be -6. 6 + 12 = 18
    See https://en.wikipedia.org/wiki/Time_zone for a list of the GMT/UTC timezone offsets
    */
    config.time.ntp_server = F("pool.ntp.org,time.nist.gov");
    config.time.gmt_offset = 3;
    config.time.day_light_offset = 0;

    /* The full message sending logs can now save to file */
    /* Since v3.0.4, the sent logs stored in smtp.sendingResult will store only the latest message logs */
    // config.sentLogs.filename = "/path/to/log/file";
    // config.sentLogs.storage_type = esp_mail_file_storage_type_flash;



    /* Set the message headers */
    /* Declare the message class */
```

```
    SMTP_Message message;
    message.sender.name = F("Emergency Alerts IIT Goa");
    message.sender.email = AUTHOR_EMAIL;
    message.subject = F("Emergency in Lab 1");
    message.addRecipient(RECIPIENT_EMAIL[i][0], RECIPIENT_EMAIL[i][1]);

    //String htmlMsg = "<p>Hi! there is an <span style=\"color:#ff0000;\">Emergency in the Lab
1</span></p><p>The message was sent via ESP device.</p>";
    String htmlMsg = "<p>This mail is to inform you that there is an <span
style=\"color:#ff0000;\"><b>Emergency</b></span> in the <b>Lab 1</b></p><p>Please provide help as soon as
possible</p><p style=\"color: gray;\"><em>Please note that this is computer generated mail<br>Don't reply
back.</em></p>";
    message.html.content = htmlMsg;
    message.html.charSet = F("us-ascii");
    message.html.transfer_encoding = Content_Transfer_Encoding::enc_7bit;
    message.priority = esp_mail_smtp_priority::esp_mail_smtp_priority_low;
    message.addHeader(F("Message-ID: alerts.iitgoa@gmail.com>"));

    if (!smtp.connect(&config))
    {
      MailClient.printf("Connection error, Status Code: %d, Error Code: %d, Reason: %s\n", smtp.statusCode(),
smtp.errorCode(), smtp.errorReason().c_str());
      return;
    }

    if (!smtp.isLoggedIn())
    {
      Serial.println("Error, Not yet logged in.");
    }
    else
    {
      if (smtp.isAuthenticated())
        Serial.println("Successfully logged in.");
      else
        Serial.println("Connected with no Auth.");
    }

    /* Start sending Email and close the session */
    if (!MailClient.sendMail(&smtp, &message))
      MailClient.printf("Error, Status Code: %d, Error Code: %d, Reason: %s\n", smtp.statusCode(),
smtp.errorCode(), smtp.errorReason().c_str());

    // to clear sending result log
    // smtp.sendingResult.clear();

    MailClient.printf("Free Heap: %d\n", MailClient.getFreeHeap());
  }
  lcd.clear();    // After the mail is been sent, it clears the previous message and writes again, that email
is been sent in the LCD
  lcd.print("Email is been");
  lcd.setCursor(0, 1);
  lcd.print("sent");
}
```

- Security office code

```
/*
 * This ESP32 code is created by esp32io.com
 *
 * This ESP32 code is released in the public domain
 *
 * For more detail (instruction and wiring diagram), visit https://esp32io.com/tutorials/communication-
between-two-esp32
 */

// ESP32 #2: TCP SERVER + AN LED
#include <Arduino.h>
#if defined(ESP32) || defined(ARDUINO_RASPBERRY_PI_PICO_W)
#include <WiFi.h>
#elif defined(ESP8266)
#include <ESP8266WiFi.h>
#elif __has_include(<WiFiNINA.h>)
#include <WiFiNINA.h>
#elif __has_include(<WiFi101.h>)
#include <WiFi101.h>
#elif __has_include(<WiFiS3.h>)
#include <WiFiS3.h>
#endif
#include <LiquidCrystal.h>


LiquidCrystal lcd(D8, D5, D4, D3, D2, D1);


#define LED_PIN  D7 // ESP32 pin GPIO18 connected to LED
#define SERVER_PORT 4080

const char* ssid = "Utsav-PC";     // CHANGE TO YOUR WIFI SSID
const char* password = "utsavbansal"; // CHANGE TO YOUR WIFI PASSWORD

WiFiServer TCPserver(SERVER_PORT);

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);

  lcd.begin(16, 2);
  lcd.clear();

  Serial.println("ESP32 #2: TCP SERVER + AN LED");

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  // Print your local IP address:
  Serial.print("ESP32 #2: TCP Server IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("ESP32 #2: -> Please update the serverAddress in ESP32 #1 code");

  // Start listening for a TCP client (from ESP32 #1)
  TCPserver.begin();
}

void loop() {
  // Wait for a TCP client from ESP32 #1:
```

```
  WiFiClient client = TCPserver.available();

  if (client) {
    // Read the command from the TCP client:
    char command = client.read();
    Serial.print("ESP32 #2: - Received command: ");
    Serial.println(command);

    if (command == '1'){
      digitalWrite(LED_PIN, HIGH); // Turn LED on
        lcd.print("Emergency in");
        lcd.setCursor(0, 1);
        lcd.print("Electrical Lab");
    }

    else if (command == '0'){
      digitalWrite(LED_PIN, LOW);  // Turn LED off
      lcd.clear();
      client.stop();
  }
}
}
```

PROGRAMMING ASPECTS

The primary focus was on integrating various functionalities, including:

1. Input Signal Handling: We programmed the Arduino to effectively handle input signals from the switch, which served as a trigger for initiating the emergency protocol.
2. NodeMCU Integration via TCPClient: Utilizing the TCPClient library, we established a connection between the NodeMCU modules. This connection facilitated communication between the devices, enabling the transmission of commands and data.
3. Command Transmission for Device Control: Through the established connection, we implemented code to send commands from to the NodeMCU modules. These commands were responsible for controlling the ON/OFF states of both the switch and the buzzer, allowing for the activation and deactivation of these components as required during emergency situations.
4. Integration with SMTP Server for Email Sending: Additionally, we incorporated code to interact with an SMTP server for sending emails. This functionality was crucial for alerting the security room about the emergency situation. Upon receiving the input signal and initiating the emergency protocol, the Arduino triggered the SMTP server to send an email containing relevant information to the designated email address of the security room.

By integrating these programming aspects, we achieved a comprehensive solution for managing emergency situations, encompassing both hardware control and communication with external systems.

CHALLENGES FACED

- We initially encountered unfamiliarity with NodeMCU, prompting us to seek learning resources.
- YouTube tutorials became our primary source of guidance to understand NodeMCU functionalities.
- Understanding the operations of SMTP server and TCP Client also required considerable time and effort.
- Identifying a cost-free platform suitable for message transmission posed a challenge.
- Despite initial hurdles, we persevered and successfully resolved the issue.
- Diligent research and troubleshooting were pivotal in acquiring necessary knowledge.
- Leveraging online tutorials provided practical insights and tips to overcome obstacles.
- Our commitment to cost-effective solutions led us to diligently search for a free messaging platform.
- Overcoming initial barriers underscored the importance of resourcefulness and persistence.
- Through online resources and problem-solving skills, we gained proficiency in NodeMCU implementation.
- This experience broadened our technical expertise and reinforced the value of persistence in overcoming challenges.

KEY OUTCOMES

Our lab report presents the successful development and implementation of an integrated IoT-based emergency response system aimed at providing swift and effective alerts during critical situations. Key outcomes include:

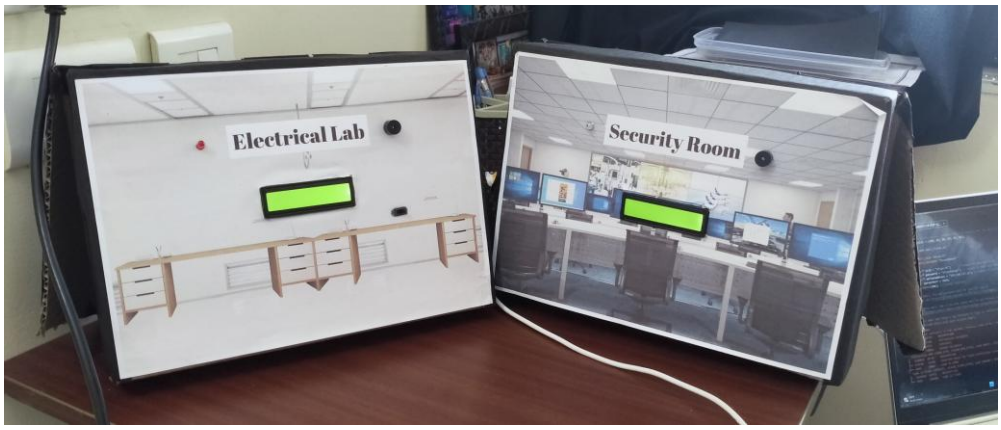
1. Effective Integration of IoT Technology: By incorporating IoT devices such as the NodeMCU alongside traditional hardware components, we have bridged the digital-physical gap, enhancing the responsiveness and reliability of the system.
2. Automated Alert Triggering: The system's ability to automatically trigger alerts in response to various emergencies, including fire outbreaks and medical emergencies, ensures a prompt and coordinated response from personnel or relevant authorities.
3. Seamless Communication: Through the integration of SMTP server and TCP Client operations, we have facilitated seamless communication, allowing for the timely transmission of alerts and messages to the security office.
4. Resourcefulness and Persistence: Overcoming initial challenges with NodeMCU unfamiliarity highlighted the importance of resourcefulness and perseverance in navigating technical hurdles, ultimately leading to a proficient understanding of the system's implementation.

In summary, our lab report showcases the successful development of an IoT-based emergency response system, emphasizing its effectiveness in alerting and communicating during critical situations while highlighting the importance of resourcefulness and persistence in overcoming technical challenges.

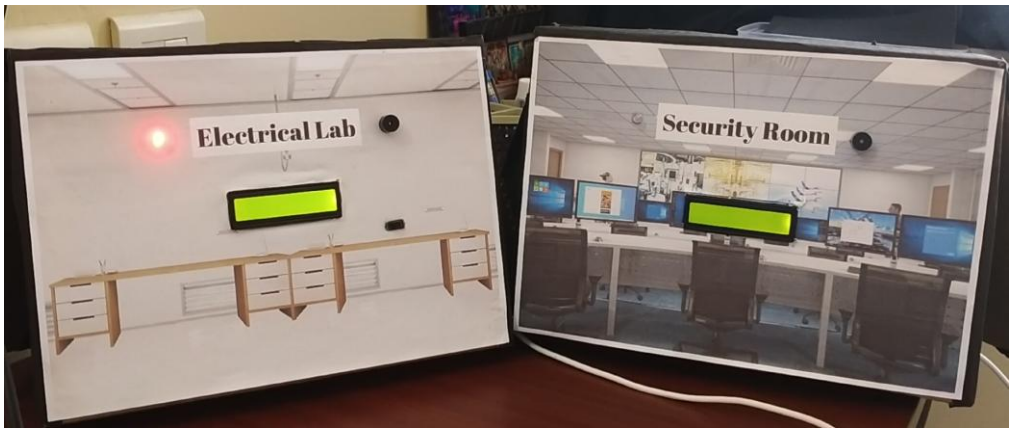
VIDEO & IMAGES

Video Link: Google Drive

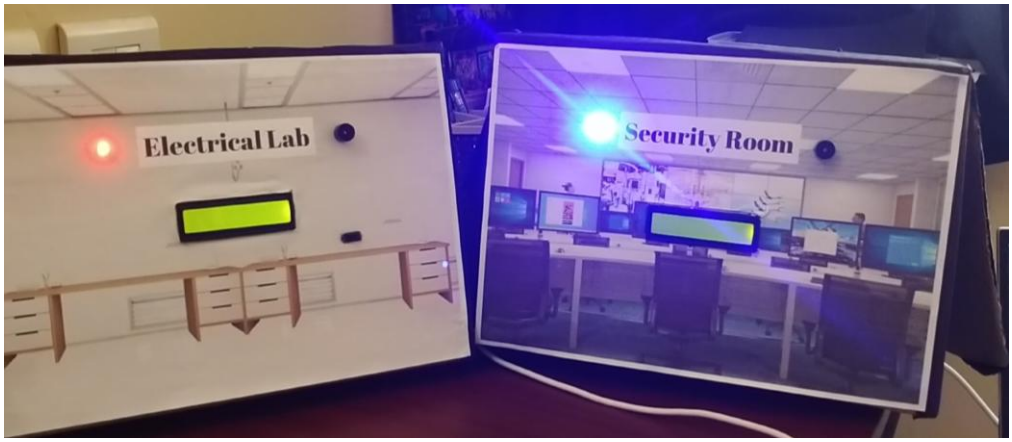
- When switch turn OFF



- Immediately after switch ON



- After Sending Email



REFERENCE LINKS

- https://randomnerdtutorials.com/projects-esp8266/
- https://youtu.be/YN522_npNqs?si=MRwNCpdJKPfmFOXx
- https://youtu.be/YN522_npNqs?si=MRwNCpdJKPfmFOXx
- https://youtu.be/YN522_npNqs?si=MRwNCpdJKPfmFOXx
- https://lededitpro.com/send-email-from-esp8266-nodemcu-via-smtp-server/
- https://www.youtube.com/watch?v=N_xP0Yw95C8

FUTURE DEVELOPMENT

- Further development and optimization of our project can significantly reduce losses and damages caused by delays in emergency response.
- Future iterations of the system can seamlessly connect various classes and departments within our college to the security office, enhancing overall coordination and response efficiency.
- The system's utility extends beyond college campuses and can be adapted for use in any educational institution or workplace as a comprehensive security protocol.
- With a compact and efficient system design, there is potential to commercialize our project by offering it to households and institutes seeking enhanced security measures.

++

THANKYOU