**CENTRO PAULA SOUZA**

**ETEC TENENTE AVIADOR GUSTAVO KLUG**

**Curso Técnico em Eletrônica**

**Allan Kleiton da Silva**

**Ednelson Emanuel dos Santos Gonçalves Fortes**

**Isabella Vitória Oliveira Longuini**

# ZELADOR VIRTUAL

**Pirassununga**

**2024**

**Allan Kleiton da Silva**

**Ednelson Emanuel dos Santos Gonçalves Fortes**

**Isabella Vitória Oliveira Longuini**

# ZELADOR VIRTUAL

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Eletrônica da Etec Tenente Aviador Gustavo Klug, orientado pelo Professor Eduardo De Franceschi como requisito parcial para obtenção do título de Técnico em Eletrônica.

**Pirassununga**

**2024**

## RESUMO

O sistema denominado ***Zelador Virtual*** fará o trabalho de monitoria predial dos sistemas a fim de diminuir erros humanos, proporcionando facilidade através de medições automáticas e análises 24 horas por dia, com o objetivo de fornecer dados das funcionalidades do prédio em tempo real como: bomba d'água, nível e temperatura, abertura e fechamento de portão social e de veículos.



**Palavras-Chave:** Zelador, Monitoria, Sistemas.

## ABSTRACT

The ***Virtual Janitor*** is a system that will carry out the work of building monitoring systems in order to reduce human errors and provide ease through automatic measurements and analyzes 24 hours a day, with the aim of providing data on the building's functionalities in real time , such as: water pump, level and temperature, opening and closing of social vehicle gates.



**Keywords:** Janitor, Monitoring, Systems.

**SUMÁRIO**

## 1 INTRODUÇÃO

O sistema ***Zelador Virtual*** vem da necessidade das pessoas de monitorar e verificar processos prediais que antes eram feitos de forma manual. Hoje com os recursos tecnológicos existentes, há a possibilidade de automatizar todos esses processos.

No processo manual de gestão de recursos prediais há muitas falhas humanas, a qual, se resultam em perdas significativas de materiais e tempo, com isso surgiu a necessidade de administrar e averiguar esses recursos de forma automática e evitar erros humanos.

Para que haja tranquilidade no processo de gestão e administração de recursos prediais há a necessidade de verificar e automatizar colocando sensores de precisão nos locais a serem analisados, seja na caixa d'água para o nível d'água, para o portão, para controle de acesso de carros e pessoas, ou na piscina com a temperatura.

O objetivo do Zelador Virtual é trazer a praticidade para o dia a dia de um condomínio, realizando as medições e alertando quando há erros, ou quando algo esteja fora do normal, para que haja recursos e preparação do funcionário de corrigir os erros alertados pelo sistema automatizado do Zelador Virtual.

No mundo globalizado através da tecnologia há a necessidade de facilitar a vida e o cotidiano das pessoas para que evite erros e desperdícios de recursos, trazendo praticidade e conforto na administração dos pontos a serem verificados.

Com base em pesquisas em campo, sendo ela uma conversa informal, concluímos que havia a necessidade de trazer facilidade e conforto na hora de averiguar os componentes do prédio.

O trabalho foi desenvolvido primeiramente na parte teórica a qual expomos ideias iniciais do trabalho, e através de pesquisas e leituras feitas em alguns sites como: *“Zeladoria Virtual é a novidade da Embracon na Expo Condomínio”* e “*Came do Brasil”,* no dia 25 de setembro de 2024, a qual os sites abordaram o tema de modernização e automatização de funções prediais no dia a dia de um zelador procurando a virtualização total das funções, tais como identificar e resolver problemas em um ambiente predial com base em tecnologia desenvolvida através da eletrônica com base na alta demanda desse serviço.

Também foi elaborada uma pesquisa de opinião sobre o tema gerando

respostas e bases para a formulação de um projeto mais robusto com o objetivo de atingir a vida das pessoas.

Posteriormente foi levantado os preços dos componentes, sensores, placas e materiais necessários, para o desenvolvimento de uma placa que execute as funções de analisar e manipular as atribuições exigidas do projeto.

Com as ideias em mente, começamos a aplicar no simulador chamado Isis Protheus, para simularmos situações procurando a eficiência do sistema.

Após fazermos todas as simulações necessárias iniciou-se a programação da placa e do supervisório a qual será executado o código fonte e suas diligências do projeto no hardware.

Posteriormente começamos a construir a placa e colocar todos os componentes nos seus lugares onde cada componente terá sua função para o funcionamento da placa.

Com a placa montada e o software pronto fizemos o upload do software na placa e por fim testamos.

Ao fazermos todos os testes fizemos verificações de funcionamento na prática para assegurar a eficácia do funcionamento do aparelho em si.

Com o projeto já montado e funcionando corretamente ele poderá ser exposto e também usado, pois a eficácia e o funcionamento dele beneficia os seus usuários.

# 2 DESENVOLVIMENTO

Esse projeto foi desenvolvido com base na demanda a qual há a necessidade de monitoria predial com base na ascensão da automação residencial a fim de facilitar a administração dos recursos prediais disponíveis.

Com isso, criamos um dispositivo que irá administrar sensores no prédio que será instalado, usando dispositivos programáveis para o uso predial administrativo.

No início criamos a ideia e aplicamos em um sistema de simulação chamado Isis Protheus para testarmos o que cada componente iria fazer na placa e o que cada placa feita irá medir e analisar, no total serão três placas que irão ser administradas, cada uma das três placas com o seu objetivo.

As placas irão passar as informações para um servidor MQTT a qual irá administrar as informações transformando em resultados visíveis em tela.

## 2.1 Desenvolvimento do código do microcontrolador

### 2.1.1 Introdução

Este código foi desenvolvido para o microcontrolador Wemos D1 mini com a finalidade de configurar a conexão Wi-Fi e integrar o dispositivo a uma rede MQTT. Ele permite que o ESP8266 atue como um ponto de acesso, permitindo que o usuário configure as credenciais Wi-Fi através de uma página web. As credenciais são armazenadas na memória EEPROM, possibilitando a reconexão automática do dispositivo a essa rede.

Após conectar-se à rede, o código estabelece uma comunicação MQTT, permitindo a publicação de mensagens relacionadas ao estado de um botão físico. A reconexão automática ao servidor MQTT é garantida, caso a conexão seja perdida. Esse sistema é ideal para aplicações de automação e controle remoto, com potencial de expansão para outros dispositivos e funcionalidades.

Abaixo será explicado todas as funcionalidades do código.

### 2.1.2 Inclusão de bibliotecas

O código inicia com a inclusão de bibliotecas essenciais para o funcionamento do sistema baseado no microcontrolador ESP8266. As bibliotecas utilizadas são

descritas a seguir:

```
#include <ESP8266WiFi.h>     // Biblioteca para controle Wi-Fi
#include <ESP8266WebServer.h> // Biblioteca para servidor web
#include <EEPROM.h>          // Biblioteca para memória EEPROM
#include <PubSubClient.h>    // Biblioteca para protocolo MQTT
```

A biblioteca **ESP8266WiFi.h** permite conectar o ESP8266 a redes Wi-Fi e gerenciar a comunicação sem fio.

A **ESP8266WebServer** cria um servidor web no dispositivo, permitindo controlar ou visualizar dados via navegador.

A **EEPROM** acessa a memória não volátil para armazenar dados persistentes, como configurações de rede.

A **PubSubClient** implementa o protocolo MQTT, facilitando a troca de mensagens leves entre dispositivos IoT, ideal para controle remoto e monitoramento.

### 2.1.3 Definições e Configurações Iniciais

```
#define LED_INDICADOR D4
#define PINO_BOTAO D1
#define LED_BOTAO D2
String TOPICO_MQTT;

// Página HTML da interface de configuração
const char MAIN_page[] PROGMEM = R"=====(
// Conteúdo HTML omitido para brevidade
)=====";

// Configurações do ponto de acesso
const char *NOME_AP = "Zelador Virtual AP";
const char *SENHA_AP = "zeladorAP";
```

```
// Configurações MQTT
const char *SERVIDOR_MQTT = " 10.0.0.126";

WiFiClient clienteWiFi;
PubSubClient clienteMQTT(clienteWiFi);
ESP8266WebServer servidorHTTP(80);

// Variáveis globais
bool ultimoEstadoBotao = HIGH;
```

O código começa com a definição de pinos, variáveis e configurações essenciais para o funcionamento do sistema

#### 2.1.3.1 Definição de Pinos e Variáveis Globais

`#define LED_INDICADOR D4`: Define o pino D4 para controle de um LED indicador.

`#define PINO_BOTAO D1`: Define o pino D1 para o botão de entrada.

`#define LED_BOTAO D2`: Define o pino D2 para o led do botão.

`String TOPICO_MQTT;`: Variável que armazena o tópico MQTT utilizado para comunicação.

#### 2.1.3.2 Página HTML da Interface de Configuração

`const char MAIN_page[] PROGMEM = R"=====(...);`: Define o conteúdo HTML da página de configuração, armazenando-a em memória programável (PROGMEM) para otimizar o uso de memória RAM. O conteúdo da página HTML é omitido para brevidade.


Figura 1 – Pagina para definição da rede wifi

#### 2.1.3.3 Configurações do Ponto de Acesso

`const char *NOME_AP = "Zelador Virtual AP";`: Define o nome da rede Wi-Fi criada pelo ESP8266.

`const char *SENHA_AP = "zeladorAP";`: Define a senha para a rede Wi-Fi.

#### 2.1.3.4 Configurações MQTT

`const char *SERVIDOR_MQTT = " 10.0.0.126";`: Define o servidor MQTT utilizado para comunicação entre dispositivos.

#### 2.1.3.5 Objetos e Variáveis Globais

`WiFiClient clienteWiFi;`: Objeto que gerencia a conexão Wi-Fi do ESP8266.

`PubSubClient clienteMQTT(clienteWiFi);`: Objeto para gerenciar a comunicação MQTT utilizando o cliente Wi-Fi.

`ESP8266WebServer servidorHTTP(80);`: Cria um servidor HTTP na porta 80 para fornecer a interface web.

`bool ultimoEstadoBotao = HIGH;`: Variável para armazenar o último estado do botão (HIGH indica botão não pressionado).

Esse trecho configura o ponto de acesso Wi-Fi, o servidor MQTT e os objetos necessários para a comunicação e controle do dispositivo, além de definir a estrutura básica para a interface de configuração via web.

### 2.1.4 Setup inicial

```
void setup() {
  pinMode(LED_INDICADOR, OUTPUT);
  pinMode(LED_BOTAO, OUTPUT);
  pinMode(PINO_BOTAO, INPUT_PULLUP);

  Serial.begin(115200);

  // Inicializa o modo Access Point
  inicializaPontoDeAcesso();
  inicializaServidorHTTP();
  configuraClienteMQTT();
}
```

A função `setup ()` é responsável pela configuração inicial do dispositivo, incluindo a definição dos pinos e a inicialização de componentes essenciais, como o ponto de acesso, servidor HTTP e cliente MQTT.

#### 2.1.4.1 Configuração de Pinos

`pinMode (LED_INDICADOR, OUTPUT);` define o pino do LED indicador (D4) como saída.

`pinMode(LED_BOTAO, OUTPUT);` define o pino do LED do botão (D2) como saída.

`pinMode (PINO_BOTAO, INPUT_PULLUP);` configura o pino do botão (D1) como entrada com resistência pull-up interna, garantindo que o pino leia um valor estável quando o botão não estiver pressionado.

#### 2.1.4.2 Inicialização da Comunicação Serial

`Serial.begin (115200);` inicia a comunicação serial com a taxa de 115200 bps para depuração e monitoramento.

#### 2.1.4.3 Inicialização dos Componentes

`inicializaPontoDeAcesso ();` Função responsável por configurar o ponto de acesso Wi-Fi.

`inicializaServidorHTTP ();` Função que inicializa o servidor HTTP para interagir com a interface web.

`configuraClienteMQTT ();` Função que configura o cliente MQTT para comunicação com o broker.

Esse trecho configura os pinos, inicializa a comunicação serial e prepara os componentes essenciais para a operação do sistema, como o ponto de acesso Wi-Fi, o servidor HTTP e o cliente MQTT.

### 2.1.5 Inicialização do Access Point

```
void inicializaPontoDeAcesso() {
  WiFi.softAP(NOME_AP, SENHA_AP);
  Serial.println("Access Point configurado");
  Serial.print("Endereço IP: ");
  Serial.println(WiFi.softAPIP());
}
```

A função `inicializaPontoDeAcesso()` é responsável por configurar o ESP8266 como um ponto de acesso Wi-Fi, permitindo que outros dispositivos se conectem a ele para configuração ou interação.

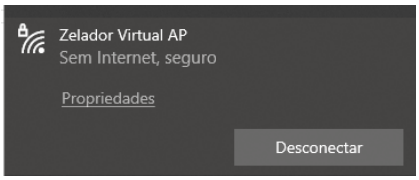



Figura 2 – Rede wifi gerada

#### 2.1.5.1 Configuração do Ponto de Acesso

`WiFi.softAP(NOME_AP, SENHA_AP);`: Inicializa o ESP8266 como um ponto de acesso Wi-Fi com o nome da rede `(NOME_AP)` e a senha `(SENHA_AP)`. Isso cria uma rede Wi-Fi à qual dispositivos podem se conectar.

#### 2.1.5.2 Exibição de Informações no Monitor Serial

`Serial.println("Access Point configurado");`: Exibe uma mensagem indicando que o ponto de acesso foi configurado com sucesso.

`Serial.print("Endereço IP: ");`: Exibe o endereço IP do ponto de acesso configurado.

`Serial.println(WiFi.softAPIP());`: Imprime o endereço IP do ponto de acesso na porta serial para que o usuário possa visualizar.

Esse trecho configura o ESP8266 para funcionar como ponto de acesso Wi-Fi, permitindo que dispositivos se conectem a ele para configurar a rede ou interagir com o sistema. A informação do endereço IP é exibida para facilitar o acesso.

```
Endereço IP: 192.168.4.1
Servidor HTTP iniciado
Tópico MQTT: PZL/social
```

Figura 3 – Informações do monitor serial

### 2.1.6 Loop Principal

```
void loop() {
  servidorHTTP.handleClient();

  // Verifica status Wi-Fi
  if (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_INDICADOR, HIGH);
    return;
  }

  // Mantém conexão MQTT ativa
  if (!clienteMQTT.connected()) {
    reconectaMQTT();
  }
  clienteMQTT.loop();
```

```
  // Monitora eventos no botão
  verificaBotao();
}
```

A função `loop()` executa continuamente as principais tarefas do sistema, garantindo a interação com o servidor HTTP, a estabilidade da conexão Wi-Fi e MQTT, além da monitoria do botão.

#### 2.1.6.1 Atendimento ao Servidor HTTP

`servidorHTTP.handleClient();`: Processa requisições recebidas pelo servidor HTTP, permitindo interação via navegador.

#### 2.1.6.2 Verificação da Conexão Wi-Fi

`if (WiFi.status ()! = WL_CONNECTED) {...}`: Verifica se o dispositivo está conectado à rede Wi-Fi. Caso a conexão seja perdida, o LED indicador é aceso (**`HIGH`**) e o restante do loop é interrompido.

#### 2.1.6.3 Manutenção da Conexão MQTT

`if (! clienteMQTT.connected()) {reconectaMQTT ();}`: Verifica se a conexão com o broker MQTT está ativa. Em caso de desconexão, chama a função `reconectaMQTT ()` para restabelecer a comunicação.

`clienteMQTT.loop ();` mantém o cliente MQTT operante, processando mensagens recebidas ou enviadas.

#### 2.1.6.4 Monitoria do Botão

`verificaBotao ();` monitora o estado do botão para detectar interações do usuário, como acionamentos ou comandos específicos.

Esse trecho assegura o funcionamento contínuo do sistema, gerenciando a comunicação via HTTP e MQTT, além de monitorar o estado da conexão Wi-Fi e a interação do usuário através do botão.

### 2.1.7 Servidor HTTP

```
void inicializaServidorHTTP() {
```

```
  servidorHTTP.on("/", exibePaginaHTML);
  servidorHTTP.on("/action_new_connection", trataNovaConexao);
  servidorHTTP.on("/action_previous_connection", conectaRedeSalva);
  servidorHTTP.begin();
  Serial.println("Servidor HTTP iniciado");
}
void exibePaginaHTML() {
  String pagina = MAIN_page;
  servidorHTTP.send(200, "text/html", pagina);
}
```

A função `inicializaServidorHTTP ()` configura o servidor HTTP para gerenciar a interface web e as rotas de comunicação.

#### 2.1.7.1 Configuração de Rotas

`servidorHTTP.on("/", exibePaginaHTML);` associa a rota principal (/) à função `exibePaginaHTML ()`, que exibe a página HTML de configuração.

`servidorHTTP.on("/action_new_connection", trataNovaConexao);` define a rota para configurar uma nova conexão Wi-Fi.

`servidorHTTP.on("/action_previous_connection", conectaRedeSalva);` define a rota para conectar a uma rede Wi-Fi previamente salva.

#### 2.1.7.2 Inicialização do Servidor

`servidorHTTP.begin();` inicia o servidor HTTP, tornando-o acessível.

`Serial.println("Servidor HTTP iniciado");` exibe no monitor serial uma mensagem de confirmação.

#### 2.1.7.3 Exibição da Página HTML

`void exibePaginaHTML ()`: Envia a página HTML armazenada na variável `MAIN_page` para o cliente conectado.

`servidorHTTP.send(200, "text/html", pagina);` responde à solicitação HTTP com o código de sucesso 200 e o conteúdo da página HTML.

Esse trecho configura o servidor HTTP, associando rotas específicas às funcionalidades do sistema e exibindo a interface web para o usuário interagir e configurar a rede Wi-Fi.

### 2.1.8 Conexão Wi-Fi e salvar credenciais na EEPROM

As funções desta seção gerenciam a conexão a redes Wi-Fi, salvam as credenciais na memória EEPROM e possibilitam a reconexão automática a redes previamente configuradas.

#### 2.1.8.1 Tratamento de Nova Conexão Wi-Fi

```
void trataNovaConexao() {
  String ssidSelecionado = servidorHTTP.arg("ssid");
  String senhaSelecionada = servidorHTTP.arg("password");

  if (!ssidSelecionado.isEmpty() && !senhaSelecionada.isEmpty()) {
    conectaRedeWiFi(ssidSelecionado, senhaSelecionada);
  } else {
    servidorHTTP.send(400, "text/plain", "SSID ou senha não fornecidos.");
  }
}
```

A função `trataNovaConexao()` gerencia o recebimento e o tratamento das credenciais de Wi-Fi enviadas via HTTP:

##### 2.1.8.1.1 Recebimento das Credenciais

Obtém as credenciais de Wi-Fi (SSID e senha) da requisição HTTP através dos métodos `servidorHTTP.arg("ssid")` e `servidorHTTP.arg("password")`.

##### 2.1.8.1.2 Verificação de Dados Válidos

Se **SSID** e **senha** não estiverem vazios, chama a função `conectaRedeWiFi()`

para tentar conectar à rede Wi-Fi usando as credenciais fornecidas.

##### 2.1.8.1.3 Resposta de Erro

Se qualquer um dos campos estiver vazio, a função envia uma resposta HTTP **400** (Bad Request) informando que **SSID ou senha não foram fornecidos**.

Essa função é responsável por lidar com a configuração da rede Wi-Fi via interface HTTP, garantindo que as credenciais sejam fornecidas corretamente e realizando a conexão com a rede.

### 2.1.8.2 Conexão a uma Rede Wi-Fi

```
void conectaRedeWiFi(const String &ssid, const String &senha) {
  WiFi.begin(ssid.c_str(), senha.c_str());
  Serial.printf("Tentando conectar a %s...\n", ssid.c_str());
  Serial.printf("Utilizando a senha %s...\n", senha.c_str());

  for (int i = 0; i < 15; ++i) {
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("Conexão estabelecida.");
      salvaCredenciaisEEPROM(ssid, senha);
      WiFi.softAPdisconnect(true);
      digitalWrite(LED_INDICADOR, LOW);
      servidorHTTP.send(200, "text/html", "<p>Conexão bem-
sucedida!</p>");
      return;
    }
    delay(500);
  }
  servidorHTTP.send(200, "text/html", "<p>Falha na conexão.</p>");
}
```

A função `conectaRedeWiFi ()` é responsável por conectar o dispositivo a uma

rede Wi-Fi especificada, utilizando o SSID e a senha fornecidos.

#### 2.1.8.2.1 Início da Conexão

`WiFi.begin(ssid.c_str (), senha.c_str ());` inicia o processo de conexão com a rede Wi-Fi utilizando o SSID e a senha fornecidos.

#### 2.1.8.2.2 Tentativa de Conexão

Executa um laço de até 15 tentativas, verificando o estado da conexão:

`if (WiFi.status() == WL_CONNECTED)`: Se a conexão for bem-sucedida:

Salva as credenciais na EEPROM através da função `salvaCredenciaisEEPROM ().`

Desativa o ponto de acesso `(WiFi.softAPdisconnect(true));`

Apaga o LED indicador `(digitalWrite (LED_INDICADOR, LOW))`;

Envia uma mensagem de sucesso ao cliente HTTP.

Caso contrário, aguarda 500 ms entre as tentativas.

#### 2.1.8.2.3 Falha na Conexão

Após 15 tentativas sem sucesso, envia ao cliente HTTP uma mensagem indicando a falha.

#### 2.1.8.2.4 Mensagens de Depuração

Utiliza `Serial.printf()` para exibir o progresso da conexão e os dados utilizados (SSID e senha) no monitor serial.

Essa função estabelece a conexão com a rede Wi-Fi, gerencia o comportamento em caso de sucesso ou falha e armazena as credenciais para uso futuro.

### 2.1.8.3 Salvamento de Credenciais na EEPROM

```
void salvaCredenciaisEEPROM(const String &ssid, const String &senha){
  EEPROM.begin(98);

  EEPROM.write(0, ssid.length());
```

```
  for (int i = 0; i < ssid.length(); ++i) {
    EEPROM.write(i + 2, ssid[i]);
  }

  EEPROM.write(1, senha.length());
  for (int i = 0; i < senha.length(); ++i) {
    EEPROM.write(i + 2 + ssid.length(), senha[i]);
  }

  EEPROM.commit();
  EEPROM.end();
}
```

A função `salvaCredenciaisEEPROM ()` armazena o SSID e a senha de uma rede Wi-Fi na memória EEPROM, permitindo reconexões automáticas futuras.

#### 2.1.8.3.1 Inicialização da EEPROM

`EEPROM.begin (98);` inicia a EEPROM com 98 bytes de espaço, permitindo operações de leitura e escrita.

#### 2.1.8.3.2 Armazenamento do SSID

`EEPROM.write (0, ssid.length ());` salva o comprimento do SSID na posição inicial da EEPROM.

Laço `for`: Escreve os caracteres do SSID, começando na posição 2.

#### 2.1.8.3.3 Armazenamento da Senha

`EEPROM.write(1, senha.length ());` Salva o comprimento da senha na segunda posição da EEPROM.

Laço `for`: Escreve os caracteres da senha, logo após o SSID armazenado.

#### 2.1.8.3.4 Gravação Permanente e Finalização

`EEPROM.commit ();` confirma as alterações, garantindo que os dados sejam gravados permanentemente.

`EEPROM.end ();` libera a memória da EEPROM, encerrando a operação.

Essa função organiza e salva as credenciais de rede de forma estruturada na EEPROM, possibilitando a recuperação e reutilização desses dados para conexões futuras, mesmo após reinicializações do dispositivo.

#### 2.1.8.4 Conexão com Rede Wi-Fi Salva

```
void conectaRedeSalva() {
  EEPROM.begin(98);

  int tamanhoSSID = EEPROM.read(0);
  int tamanhoSenha = EEPROM.read(1);

  String ssid = "", senha = "";
  for (int i = 0; i < tamanhoSSID; ++i) {
    ssid += char(EEPROM.read(i + 2));
  }
  for (int i = 0; i < tamanhoSenha; ++i) {
    senha += char(EEPROM.read(i + 2 + tamanhoSSID));
  }

  EEPROM.end();
  conectaRedeWiFi(ssid, senha);
}
```

A função `conectaRedeSalva ()` recupera as credenciais de rede Wi-Fi armazenadas na memória EEPROM e tenta estabelecer a conexão com a rede correspondente.

##### 2.1.8.4.1 Inicialização da EEPROM

`EEPROM.begin(98);` inicia a memória EEPROM para permitir operações de leitura.

#### 2.1.8.4.2 Leitura do Comprimento das Credenciais

`EEPROM.read(0);` lê o comprimento do SSID armazenado na posição inicial da EEPROM.

`EEPROM.read(1);` lê o comprimento da senha armazenada na posição seguinte.

#### 2.1.8.4.3 Recuperação das Credenciais

Laço `for` para o SSID: Constrói a string `ssid` lendo os caracteres armazenados a partir da posição `2`.

Laço `for` para a senha: Constrói a string `senha` lendo os caracteres armazenados após o SSID.

#### 2.1.8.4.4 Finalização da Leitura

`EEPROM.end ();` libera a memória EEPROM, encerrando as operações de leitura.

#### 2.1.8.4.5 Conexão com a Rede

`conectaRedeWiFi (ssid, senha);` utiliza as credenciais recuperadas para tentar conectar à rede correspondente.

Essa função automatiza a conexão com redes Wi-Fi previamente configuradas, buscando as credenciais salvas na EEPROM e estabelecendo a conexão de forma transparente para o usuário.

### 2.1.9 Configuração MQTT

#### 2.1.9.1 Configuração cliente MQTT

```
void configuraClienteMQTT() {
  TOPICO_MQTT = "PZL/nivel" /*+ WiFi.macAddress()*/;
  clienteMQTT.setServer(SERVIDOR_MQTT, 1883);
  Serial.print("Tópico MQTT: ");
  Serial.println(TOPICO_MQTT); // Para depuração
}
```

A função `configuraClienteMQTT ()` prepara o dispositivo para comunicação com o servidor MQTT, definindo o tópico de publicação e configurando o servidor.

#### 2.1.9.1.1 Definição do Tópico MQTT

O tópico MQTT é configurado como "`PZL/nivel`", que será usado para envio de mensagens pelo dispositivo.

#### 2.1.9.1.2 Configuração do Servidor MQTT

Define o servidor MQTT e a porta padrão de comunicação (1883) com o comando:

```
clienteMQTT.setServer(SERVIDOR_MQTT, 1883);
```

#### 2.1.9.1.3 Mensagem de Depuração

O tópico configurado é exibido no monitor serial para facilitar a verificação e depuração durante o desenvolvimento.

Essa função configura os elementos básicos para a comunicação MQTT, utilizando um tópico fixo para a troca de informações entre o dispositivo e o servidor.

### 2.1.9.2 Reconexão ao Servidor MQTT

```
void reconectaMQTT() {
  String clientId = "PZL_" + WiFi.macAddress();
  Serial.println("Tentando conectar ao MQTT com Client ID: " + clientId);

  while (!clienteMQTT.connected()) {
    if (clienteMQTT.connect(clientId.c_str())) {
      Serial.println("Conectado ao MQTT");
      clienteMQTT.subscribe((TOPICO_MQTT + "/in").c_str());
    } else {
      Serial.println("Falha na conexão ao MQTT");
      delay(1000);
```

```
    }
  }
}
```

A função `reconectaMQTT ()` assegura que o dispositivo esteja sempre conectado ao servidor MQTT, realizando reconexões automáticas em caso de desconexão.

#### 2.1.9.2.1 Tentativa de Reconexão

Inicia um loop de reconexão enquanto o cliente MQTT não estiver conectado:

`clienteMQTT.connect(clientId.c_str());` Tenta estabelecer a conexão com o servidor MQTT utilizando o **Client ID** gerado.

Em caso de sucesso:

Exibe a mensagem "**Conectado ao MQTT**" no monitor serial.

Inscreve-se no tópico de entrada associado ao dispositivo (`TOPICO_MQTT + "/in"`) para receber mensagens.

Em caso de falha:

Exibe a mensagem "**Falha na conexão ao MQTT**" e aguarda 1 segundo antes de tentar novamente.

Essa função garante que o dispositivo mantenha uma conexão estável com o servidor MQTT, crucial para a troca de mensagens em sistemas de IoT.

### 2.1.10 Verificação do Botão

```
void verificaBotao () {
  int estadoAtualBotao = digitalRead (PINO_BOTAO);

  if (estadoAtualBotao == HIGH && ultimoEstadoBotao == LOW) {
    clienteMQTT.publish(TOPICO_MQTT.c_str (), "Nivel Baixo!");
    digitalWrite (LED_BOTAO, HIGH);
    delay (500);
```

```
  }
  else if (estadoAtualBotao == LOW && ultimoEstadoBotao == HIGH) {
    clienteMQTT.publish(TOPICO_MQTT.c_str (), "Nivel Alto!");
    digitalWrite (LED_BOTAO, LOW);
    delay (500);
  }
  ultimoEstadoBotao = estadoAtualBotao;
}
```

A função `verificaBotao ()` monitora o estado de um botão conectado ao dispositivo e envia mensagens MQTT conforme as mudanças no estado.

#### 2.1.10.1 Leitura do Estado Atual do Botão

Utiliza `digitalRead (PINO_BOTAO)` para capturar o estado atual do botão (HIGH ou LOW).

#### 2.1.10.2 Detecção de Mudança de Estado

Compara o estado atual do botão com o último estado salvo:

**HIGH -> LOW (botão foi pressionado)**:

Publica a mensagem "**Nivel Baixo!**" no tópico MQTT configurado (`TOPICO_MQTT`).

Liga o LED indicador associado ao botão (`digitalWrite (LED_BOTAO, HIGH)`).

**LOW -> HIGH (botão foi solto):**

Publica a mensagem "**Nivel Alto!**" no tópico MQTT.

Desliga o LED indicador (`digitalWrite (LED_BOTAO, LOW)`).

#### 2.1.10.3 Atraso para Estabilidade

Introduz um pequeno atraso (`delay (500)`) para evitar falsos positivos devido a ruídos mecânicos no botão.

#### 2.1.10.4 Atualização do Estado do Botão

Armazena o estado atual do botão na variável `ultimoEstadoBotao` para

comparações futuras.

Essa função monitora eventos no botão e integra a interação física ao sistema MQTT, permitindo notificações automáticas sobre o estado do botão em tempo real.

## 3 METODOLOGIA

A presente pesquisa utilizará uma abordagem quantitativa e qualitativa para analisar os dados coletados através de um questionário online aplicado aos moradores do condomínio Recanto dos Pássaros em Pirassununga-SP. Os dados quantitativos, obtidos através das perguntas fechadas, serão analisados utilizando estatística descritiva e testes de hipóteses. Já os dados qualitativos, provenientes das respostas abertas, serão analisados através da técnica de análise de conteúdo, buscando identificar temas e categorias emergentes.

De todas as pessoas que foram entrevistadas tinham uma média de idade de 19 anos, foram entrevistados cerca de 7 moradores desse condomínio, a qual não preferiram se identificar.

Em seguida as questões aplicadas na pesquisa foram retiradas com base numa pesquisa prévia do assunto a ser tratado elaborando perguntas a qual levará a respostas e dados contundentes.

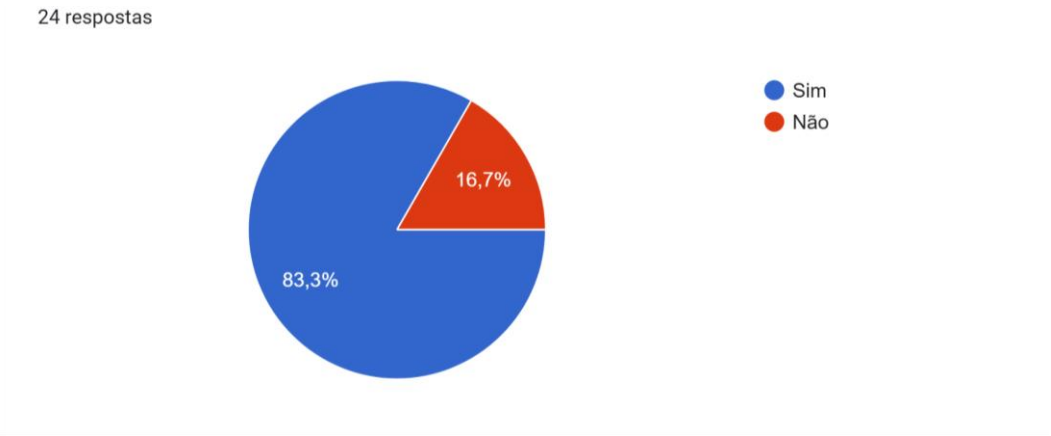
Definimos os seguintes parâmetros de 0 a 100% acima de 50% significa concordo ou resposta sim e abaixo de 50% significa não concordo ou resposta não.

O esperado da pesquisa de opinião era que ou as pessoas vissem a necessidade ou ficaram entusiasmadas com um sistema autônomo de zeladoria virtual ou não se entusiasmaram e ou não gostariam de ter esse contato com esse tipo de sistema tecnológico. Com isso foi elaborada a pesquisa onde os resultados são apresentados logo abaixo.

## 3.1 Pesquisa Qualitativa e Quantitativa.

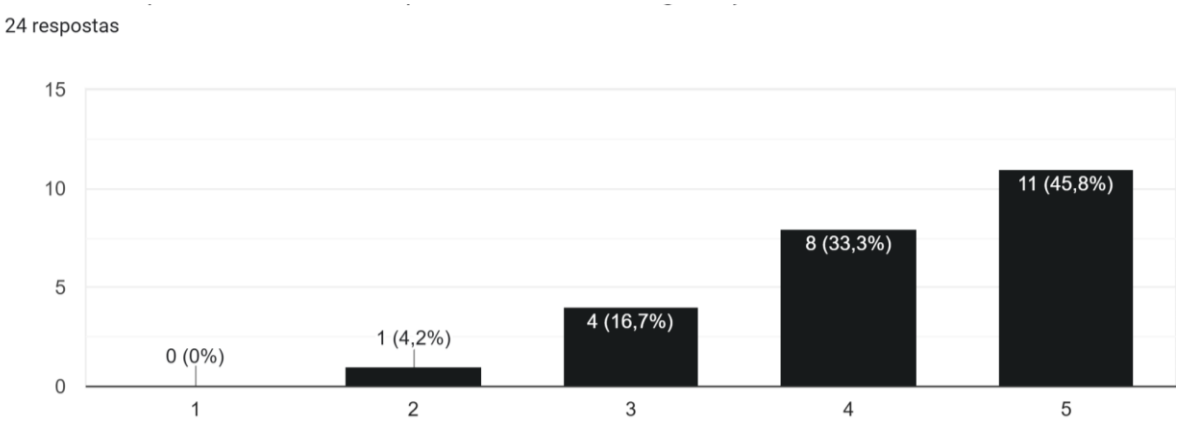
A primeira pergunta do questionário é “Você seria favorável a implementação de um zelador virtual no condomínio, que automatizaria algumas tarefas como, controle de acesso, controle de nível d'água entre outros?”, com a resposta sendo de escolha única sendo “Sim” ou “Não”, sendo sim uma afirmação e concordância com a frase da apresentada acima e não sendo uma negativa ou uma não concordância com a frase apresentada.

Obtivemos as seguintes respostas.



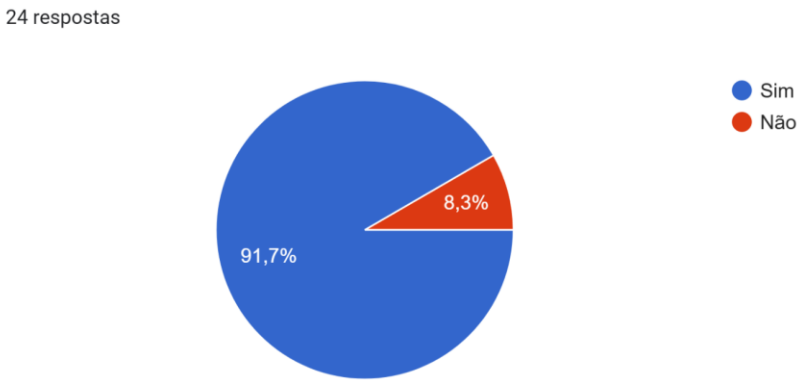

A segunda pergunta do questionário é “Você acredita que um zelador virtual poderia melhorar a segurança do condomínio?”, com a resposta sendo de escolha única sendo “Sim” ou “Não”, sendo sim uma afirmação e concordância com a frase da apresentada acima e não sendo uma negativa ou uma não concordância com a frase apresentada.

Obtivemos as seguintes respostas.



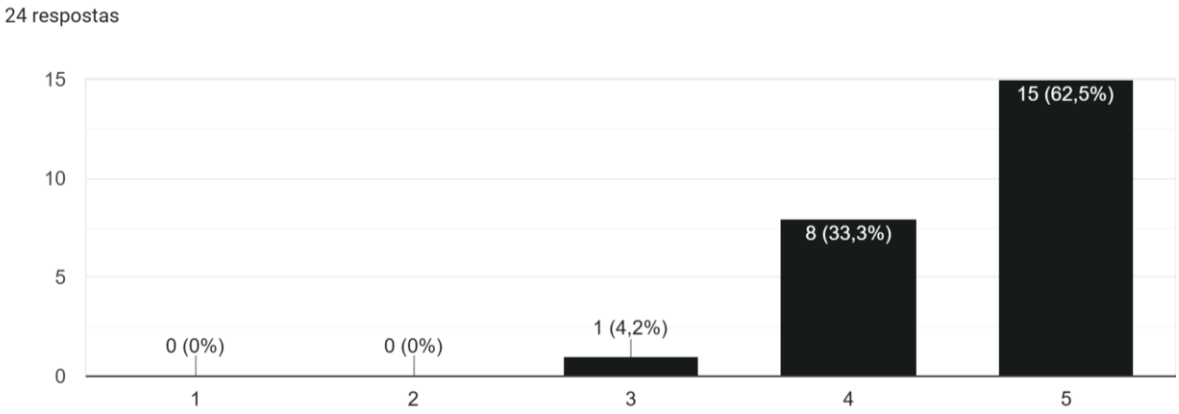

A terceira pergunta do questionário é “Você estaria disposto a usar um aplicativo ou plataforma para se comunicar com o zelador virtual e realizar solicitações?”, com a resposta sendo de escolha única sendo “Sim” ou “Não”, sendo sim uma afirmação e concordância com a frase da apresentada acima e não sendo uma negativa ou uma não concordância com a frase apresentada.

Obtivemos as seguintes respostas.



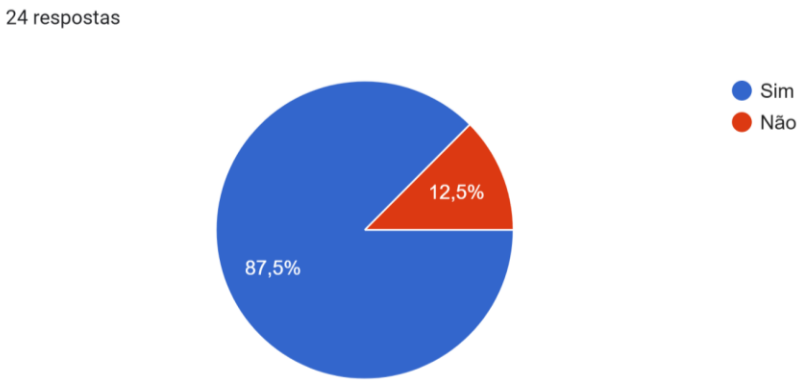

A quarta pergunta do questionário é “Você considera que a utilização de um zelador virtual agilizaria a resolução de problemas e demandas dos moradores?”, com a resposta sendo de escolha única sendo “Sim” ou “Não”, sendo sim uma afirmação e concordância com a frase da apresentada acima e não sendo uma negativa ou uma não concordância com a frase apresentada.

Obtivemos as seguintes respostas.




A quinta pergunta do questionário é “Acredita que a implementação de um zelador virtual poderia reduzir os custos do condomínio?”, com a resposta sendo de escolha única sendo “Sim” ou “Não”, sendo sim uma afirmação e concordância com a frase da apresentada acima e não sendo uma negativa ou uma não concordância com a frase apresentada.

Obtivemos as seguintes respostas.




## 3.2 Resultado da Pesquisa

A pesquisa sobre a implementação de zeladores virtuais em condomínios revela uma crescente tendência de modernização e aceitação na gestão virtual condominial.

Com os dados obtidos observamos que há aceitação na implementação desse sistema, com uma absorção das pessoas sob uma nova tecnologia em seus

condomínios, pois cerca de 8 em cada 9 pessoas têm demonstrado esse interesse com respostas positivas sobre os temas abordados.

A pesquisa demonstra que a implementação de zeladores virtuais em condomínios é uma tendência irreversível, trazendo benefícios como maior eficiência, segurança e redução de custos. No entanto, é preciso considerar os desafios e trabalhar em conjunto para garantir uma transição suave e exitosa.

## 4 CONSIDERAÇÕES FINAIS

A presente pesquisa demonstrou que a figura do Zelador Virtual pode ser promissora pois possui um grande potencial para revolucionar a gestão condominial. A alta taxa de aceitação por parte dos condôminos indica que a sociedade está pronta para adotar soluções tecnológicas que otimizem processos e melhorem a qualidade de vida.

Há a necessidade de modernização na gestão condominial e a busca por soluções mais eficientes e acessíveis.

Embora a aceitação seja alta, é importante ressaltar que alguns desafios ainda precisam ser superados, como a necessidade de investir em infraestrutura tecnológica e a capacitação dos funcionários.

No entanto, é fundamental que os desafios relacionados à implementação e à capacitação sejam superados para garantir o sucesso dessa nova ferramenta. As perspectivas futuras são animadoras, com a expectativa de que o Zelador Virtual se torne cada vez mais presente nos condomínios, transformando a forma como vivemos e interagimos com nossos espaços.

# 5 CÓDIGO DO PROJETO

```
#include <ESP8266WiFi.h>     // Biblioteca para controle Wi-Fi
#include <ESP8266WebServer.h> // Biblioteca para servidor web
#include <EEPROM.h>          // Biblioteca para memória EEPROM
#include <PubSubClient.h>    // Biblioteca para protocolo MQTT

// Definições de pinos
#define LED_INDICADOR D4
#define PINO_BOTAO D1
#define LED_BOTAO D2
String TOPICO_MQTT;

// Página para conctar rede wifi
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Configuração de Conexão</title>

  <style>
    :root {
      --primary-color: #2c3e50;
      --secondary-color: #3498db;
      --background-dark: #121212;
      --input-background: #f4f4f4;
    }
```

```
    * {
      box-sizing: border-box;
      margin: 0;
      padding: 0;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      background: linear-gradient(135deg, var(--background-dark) 0%, #1c1c1c
100%);
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      line-height: 1.6;
    }

    .network-container {
      background-color: white;
      width: 100%;
      max-width: 420px;
      padding: 40px 30px;
      border-radius: 12px;
      box-shadow: 0 15px 35px rgba(0,0,0,0.1);
      text-align: center;
    }

    .app-title {
      font-size: 2.5rem;
      margin-bottom: 20px;
      display: flex;
```

```
  justify-content: center;
  align-items: center;
}

.app-title span:first-child {
  color: var(--secondary-color);
  margin-right: 10px;
}

.app-title span:last-child {
  color: var(--primary-color);
}

.network-form {
  display: flex;
  flex-direction: column;
  gap: 15px;
}

.input-wrapper {
  position: relative;
}

.network-input {
  width: 100%;
  padding: 12px 15px;
  border: 2px solid #e0e0e0;
  border-radius: 8px;
  font-size: 1rem;
  transition: border-color 0.3s ease;
```

```
}

.network-input:focus {
  outline: none;
  border-color: var(--secondary-color);
}

.password-toggle {
  position: absolute;
  right: 10px;
  top: 50%;
  transform: translateY(-50%);
  background: none;
  border: none;
  cursor: pointer;
  color: var(--primary-color);
  opacity: 0.6;
  transition: opacity 0.3s;
}

.password-toggle:hover {
  opacity: 1;
}

.submit-btn {
  background-color: var(--secondary-color);
  color: white;
  border: none;
  padding: 12px;
  border-radius: 8px;
```

```
      font-weight: bold;
      cursor: pointer;
      transition: background-color 0.3s;
    }

    .submit-btn:hover {
      background-color: #2980b9;
    }

    .status-message {
      margin: 10px 0;
      font-size: 0.9rem;
    }

    .alternative-action {
      margin-top: 20px;
      display: flex;
      flex-direction: column;
      gap: 15px;
    }
  </style>
  <script>
    function validateNetworkConfig() {
      const networkSelect = document.querySelector('select[name="ssid"]');
      const passwordInput = document.querySelector('input[name="password"]');
      const statusArea = document.getElementById('status-area');

      if (!networkSelect.value) {
        statusArea.innerHTML = '<span style="color:red;">Selecione uma rede</span>';
```

```
        return false;
      }

      if (!passwordInput.value) {
        statusArea.innerHTML = '<span style="color:red;">Digite a senha da rede</span>';
        return false;
      }

      statusArea.innerHTML = '<span style="color:green;">Conectando...</span>';
      return true;
    }

    function togglePasswordVisibility() {
      const passwordInput = document.querySelector('input[name="password"]');
      passwordInput.type = passwordInput.type === 'password' ? 'text' : 'password';
    }
  </script>
</head>
<body>
  <div class="network-container">
    <div class="app-title">
      <span>Conexão</span>
      <span>Rápida</span>
    </div>

    <form class="network-form" action="/action_new_connection" method="post" onsubmit="return validateNetworkConfig()">
      <div class="input-wrapper">
        <input
```

```
    type="text"
    class="network-input"
    name="ssid"
    placeholder="Digite o SSID"
    maxlength="64"
    required
  >
</div>

<div class="input-wrapper">
  <input
    type="password"
    class="network-input"
    name="password"
    placeholder="Senha da Rede"
    maxlength="64"
    required
  >
  <button
    type="button"
    class="password-toggle"
    onclick="togglePasswordVisibility()"
  >
    👁
  </button>
</div>

<div id="status-area" class="status-message"></div>

<button type="submit" class="submit-btn">
```

```
        Conectar Agora
      </button>
    </form>

    <div class="alternative-action">
      <p>Ou</p>
      <button
        class="submit-btn"
        onclick="window.location.href='/action_previous_connection'"
      >
        Usar Última Conexão
      </button>
    </div>
  </div>
</body>
</html>
)=====";

// Configurações do ponto de acesso
const char *NOME_AP = "Zelador Virtual AP";
const char *SENHA_AP = "zeladorAP";

// Configurações MQTT
const char *SERVIDOR_MQTT = "10.0.0.126";

WiFiClient clienteWiFi;
PubSubClient clienteMQTT(clienteWiFi);
ESP8266WebServer servidorHTTP(80);

// Variáveis globais
```

```
bool ultimoEstadoBotao = HIGH;

void setup() {
  pinMode(LED_INDICADOR, OUTPUT);
  pinMode(LED_BOTAO, OUTPUT);
  pinMode(PINO_BOTAO, INPUT_PULLUP);

  Serial.begin(115200);

  // Inicializa o modo Access Point
  inicializaPontoDeAcesso();
  inicializaServidorHTTP();
  configuraClienteMQTT();
}

void loop() {
  servidorHTTP.handleClient();

  // Verifica status Wi-Fi
  if (WiFi.status() != WL_CONNECTED) {
    digitalWrite(LED_INDICADOR, HIGH);
    return;
  }

  // Mantém conexão MQTT ativa
  if (!clienteMQTT.connected()) {
    reconectaMQTT();
  }
  clienteMQTT.loop();
```

```
  // Monitora eventos no botão
  verificaBotao();
}

void inicializaPontoDeAcesso() {
  WiFi.softAP(NOME_AP, SENHA_AP);
  Serial.println("Access Point configurado");
  Serial.print("Endereço IP: ");
  Serial.println(WiFi.softAPIP());
}

void inicializaServidorHTTP() {
  servidorHTTP.on("/", exibePaginaHTML);
  servidorHTTP.on("/action_new_connection", trataNovaConexao);
  servidorHTTP.on("/action_previous_connection", conectaRedeSalva);
  servidorHTTP.begin();
  Serial.println("Servidor HTTP iniciado");
}

void configuraClienteMQTT() {
  TOPICO_MQTT = "PZL/nivel" /*+ WiFi.macAddress()*/;
  clienteMQTT.setServer(SERVIDOR_MQTT, 1883);
  Serial.print("Tópico MQTT: ");
  Serial.println(TOPICO_MQTT); // Para depuração
}

void exibePaginaHTML() {
  String pagina = MAIN_page;
  servidorHTTP.send(200, "text/html", pagina);
}
```

```
void trataNovaConexao() {
  String ssidSelecionado = servidorHTTP.arg("ssid");
  String senhaSelecionada = servidorHTTP.arg("password");

  if (!ssidSelecionado.isEmpty() && !senhaSelecionada.isEmpty()) {
    conectaRedeWiFi(ssidSelecionado, senhaSelecionada);
  } else {
    servidorHTTP.send(400, "text/plain", "SSID ou senha não fornecidos.");
  }
}

void conectaRedeWiFi(const String &ssid, const String &senha) {
  WiFi.begin(ssid.c_str(), senha.c_str());
  Serial.printf("Tentando conectar a %s...\n", ssid.c_str());
  Serial.printf("Utilizando a senha %s...\n", senha.c_str());

  for (int i = 0; i < 15; ++i) {
    if (WiFi.status() == WL_CONNECTED) {
      Serial.println("Conexão estabelecida.");
      salvaCredenciaisEEPROM(ssid, senha);
      WiFi.softAPdisconnect(true);
      digitalWrite(LED_INDICADOR, LOW);
      servidorHTTP.send(200, "text/html", "<p>Conexão bem-sucedida!</p>");
      return;
    }
    delay(500);
  }
  servidorHTTP.send(200, "text/html", "<p>Falha na conexão.</p>");
}
```

```
void salvaCredenciaisEEPROM(const String &ssid, const String &senha) {
  EEPROM.begin(98);

  EEPROM.write(0, ssid.length());
  for (int i = 0; i < ssid.length(); ++i) {
    EEPROM.write(i + 2, ssid[i]);
  }

  EEPROM.write(1, senha.length());
  for (int i = 0; i < senha.length(); ++i) {
    EEPROM.write(i + 2 + ssid.length(), senha[i]);
  }

  EEPROM.commit();
  EEPROM.end();
}

void conectaRedeSalva() {
  EEPROM.begin(98);

  int tamanhoSSID = EEPROM.read(0);
  int tamanhoSenha = EEPROM.read(1);

  String ssid = "", senha = "";
  for (int i = 0; i < tamanhoSSID; ++i) {
    ssid += char(EEPROM.read(i + 2));
  }
  for (int i = 0; i < tamanhoSenha; ++i) {
    senha += char(EEPROM.read(i + 2 + tamanhoSSID));
```

```
  }

  EEPROM.end();
  conectaRedeWiFi(ssid, senha);
}

void reconectaMQTT() {
  String clientId = "PZL_" + WiFi.macAddress();
  Serial.println("Tentando conectar ao MQTT com Client ID: " + clientId);

  while (!clienteMQTT.connected()) {
    if (clienteMQTT.connect(clientId.c_str())) {
      Serial.println("Conectado ao MQTT");
      clienteMQTT.subscribe((TOPICO_MQTT + "/in").c_str());
    } else {
      Serial.println("Falha na conexão ao MQTT");
      delay(1000);
    }
  }
}

void verificaBotao() {
  int estadoAtualBotao = digitalRead(PINO_BOTAO);

  if (estadoAtualBotao == HIGH && ultimoEstadoBotao == LOW) {
    clienteMQTT.publish(TOPICO_MQTT.c_str(), "Nivel Baixo!");
    digitalWrite(LED_BOTAO, HIGH);
    delay(500);
  }
  else if (estadoAtualBotao == LOW && ultimoEstadoBotao == HIGH){
```

```
    clienteMQTT.publish(TOPICO_MQTT.c_str(), "Nivel Alto!");
    digitalWrite(LED_BOTAO, LOW);
    delay(500);
  }
  ultimoEstadoBotao = estadoAtualBotao;
}
```

## *6* REFERÊNCIAS

*BARBOSA,Marco, Setor de condomínios comerciais ajuda a impulsionar investimentos em segurança privada, que pela 1ª vez, desde 2018, volta a crescer no país,* Tecnologia. Disponível em: [https://camedobrasil.com.br/setor-de-condominios-comerciais-ajuda-a-impulsionar-investimentos-em-seguranca-privada-que-pela-1a-vez-desde-2018-volta-a-crescer-no-pais/]. Acesso em 25 de Setembro de 2024

*ALMEIDA, Rafael,* Zeladoria Virtual é a novidade da Embracon na Expo Condomínio e Tecnologia. Disponível em: [https://koicomunicacao.com.br/clientes/zeladoria-virtual-e-a-novidade-da-embracon-na-expo-condominio/]. Acesso em 25 de Setembro de 2024.

MARTINS, Gabriel, ESP8266 – Salvando Credenciais Wi-Fi na EEPROM Através de um Access Point (AP). Disponível em: [https://blog.smartkits.com.br/esp8266-salvando-credenciais-wi-fi-na-eeprom-atraves-de-um-access-point-ap/#:~:text=%C3%89%20importante%20saber%20qual%20endere%C3%A7o,fun%C3%A7%C3%B5es%20de%20configura%C3%A7%C3%A3o%20de%20AP]. Acesso em 28 de setembro de 2024.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**. Informação e documentação: referências: elaboração. Rio de Janeiro, 2002.