**UNIVERSIDADE DE MARÍLIA (UNIMAR)**

**Bacharelado em Ciências da Computação**

Fábrica de Projetos – II

**Climatização Inteligente: Desenvolvendo Soluções Inovadoras em Ar Condicionado**

**Integrantes:** Alex Veloso, Breno Henrique, Caio Henrique, Cauã Santos,

Gustavo Soares, Pedro Bernardo, Leandro Poletti

**INTRODUÇÃO DO PROJETO**

A automação residencial está redefinindo a maneira como interagimos com nossos dispositivos em casa, proporcionando ambientes mais inteligentes e eficientes. Um dos aparelhos fundamentais para o conforto doméstico é o ar condicionado. No entanto, muitos sistemas de ar condicionado tradicionais apresentam restrições em termos de controles e flexibilidade, deixando os usuários em busca de soluções mais adaptáveis e personalizadas. É nesse contexto que este projeto ganha vida. Nossa proposta é criar um sistema de automação de ar condicionado baseado na plataforma Arduino, um micro controlador versátil e acessível. Este sistema é projetado para oferecer um controle avançado e altamente personalizado do sistema de climatização residencial, elevando o nível de conforto e eficiência. Além disso, ele traz consigo uma série de benefícios notáveis. Ao permitir que os usuários controlem o ar condicionado através de seus dispositivos móveis e computadores, oferecemos um novo patamar de comodidade. Isso não apenas simplifica a operação do aparelho, mas também permite que os usuários ajustem o ambiente a partir de qualquer lugar, fornecendo uma experiência verdadeiramente flexível. Além disso, este projeto abre portas para a criação de um ecossistema integrado de dispositivos domésticos inteligentes. O ar condicionado pode se comunicar e colaborar com outros dispositivos, Essa integração resulta em uma casa mais eficiente, conveniente e personalizada. Neste relatório, exploraremos detalhadamente o desenvolvimento e a implementação dessa solução, destacando como ela está redefinindo a climatização residencial e oferecendo uma série de benefícios significativos para os usuários e o ambiente doméstico.

**OBJETIVO DO PROJETO:**

O objetivo primordial deste projeto é aprimorar o controle e a eficiência do sistema de ar condicionado, por meio da integração do Arduino como plataforma central de controle. A abordagem consiste em reconfigurar o micro controlador Arduino para assumir as funções tradicionalmente atribuídas ao controle convencional do ar-condicionado, com um foco especial em aprimorar o controle remoto e automático das operações.

A transformação engloba a capacidade do Arduino de executar as tarefas fundamentais, tais como ligar e desligar o sistema de ar condicionado de forma remota, bem como ajustar com precisão a temperatura desejada e a velocidade do ventilador. Esse controle avançado e integrado permitirá que os usuários gerenciem o ambiente climatizado de maneira mais conveniente e eficiente, contribuindo para uma experiência aprimorada de conforto e economia de energia.

**JUSTIFICATIVA E PROPOSTAS CORRELATAS**

A climatização da sala de aula por meio de um aplicativo online oferece vantagens significativas, como a personalização do aprendizado, o monitoramento em tempo real da temperatura ambiente da sala de aula. A automação de sistemas de ar condicionado utilizando o Arduino como plataforma central de controle oferece uma série de benefícios e se alinha com diversas necessidades e tendências na área de climatização e automação residencial.

A iniciativa de automatizar sistemas de ar condicionado utilizando o Arduino como plataforma central de controle está alinhada com tendências significativas na área de automação residencial e Internet das Coisas (IoT). Essa abordagem inovadora pode ser contextualizada em projetos anteriores que exploram a automação de ambientes e oferecem insights valiosos sobre como a tecnologia pode melhorar a qualidade de vida e a eficiência energética.

A proposta de automatizar sistemas de ar condicionado do ambiente por meio de aplicativos online tem sido objeto de interesse em diversos projetos já existentes como: Smart Place 3.0: Gerenciamento Automático de Ar Condicionados em Ambientes Complexos, Sistema automatizado de gerenciamento de ar condicionado usa sensores e câmeras para ajustar a temperatura com base na presença de pessoas, economizando energia em ambientes complexos. Projeto Ubique: Sistema de monitoramento e controle de ar-condicionado. Este projeto visa reduzir o consumo de energia nos aparelhos de ar-condicionado do Centro de Informática, controlando-os remotamente com um sistema de baixo custo que utiliza sensores e uma placa micro controladora conectada à internet via WiFi e protocolo MQTT.

**ADERÊNCIA AS DISCIPLINAS**

Este projeto de automação de ar condicionado com Arduino demonstra uma forte aderência aos conceitos de Programação Orientada a Objetos, Estrutura de Dados e Sistemas Digitais, O projeto utiliza criação de classes e objetos para representar os componentes do sistema, como sensores, atuadores e controles de temperatura uso de estruturas de dados, como ponteiros, structs e estruturas lineares, para armazenar e manipular informações relevantes, como dados de sensores e configurações do sistema e o projeto incorpora conceitos de sistemas digitais ao utilizar o Arduino como plataforma central de controle

**PROTOTIPAÇÃO**

Materiais Utilizados

-Arduino Uno: Utilizamos o Arduino Uno como a unidade de controle central para o projeto.

-Módulo Sensor de Umidade e Temperatura DHT11: Esse sensor é utilizado para medir a temperatura e umidade ambiente.

-Protoboard e Resistores: Usados para montar os circuitos de forma organizada e para fornecer resistência elétrica quando necessário.

-Kit de Jumpers: Os jumpers são usados para conectar os componentes de forma elétrica.

-LED Emissor de Infravermelho: O LED é usado para enviar comandos infravermelhos para o ar condicionado.

-Sensor IRM e Receptor Infravermelho: O sensor IRM é usado para capturar comandos infravermelhos do controle remoto do ar condicionado.

-Módulo Transmissor de Infravermelho: Usado para enviar os comandos infravermelhos para o ar condicionado.

-Módulo Emissor de Infravermelho: Usado para emitir sinais infravermelhos que são recebidos pelo ar condicionado.

-Módulo de Interface de Display LCD para Arduino: Utilizamos um display LCD para exibir informações como temperatura e status do ar condicionado.

-Placa ESP32: Utilizada para habilitar a conectividade Wi-Fi e a integração com a IoT

**FUNCIONALIDADES:**

-Medição de Temperatura e Umidade: O sensor DHT11 mede a temperatura e umidade ambiente e exibe os valores no display LCD.

-Controle Remoto do Ar Condicionado: O sistema é capaz de capturar comandos infravermelhos do controle remoto original do ar condicionado usando o sensor IRM. Ele também envia comandos infravermelhos para o ar condicionado por meio do módulo transmissor de infravermelho.

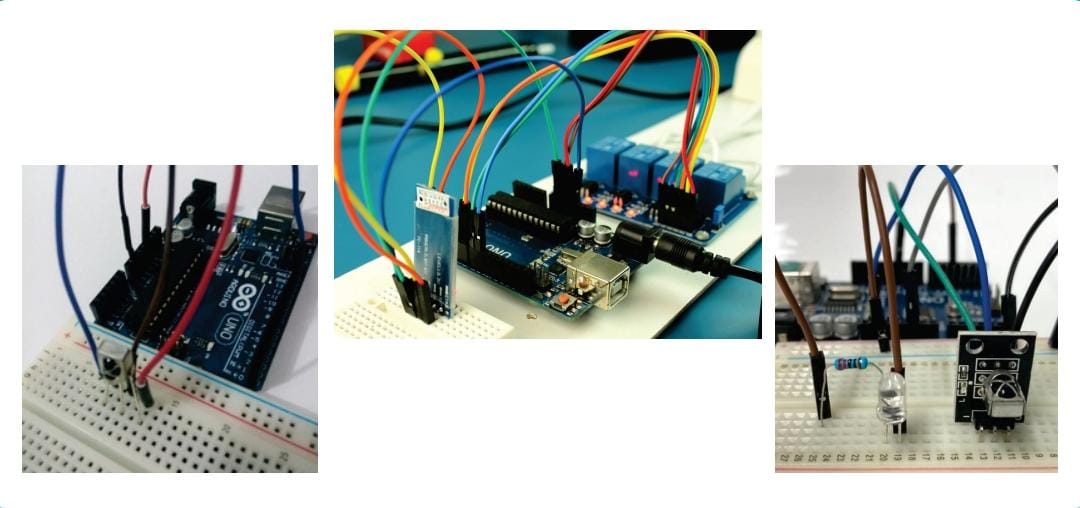
-Controle por Aplicativo IoT: Um aplicativo móvel permite que os usuários controlem o ar condicionado remotamente. Os comandos são enviados para a placa ESP32, que os encaminha para o Arduino.

-Exibição no Display: O display LCD mostra a temperatura atual, umidade e o status do ar condicionado (ligado/desligado)

**CRONOGRAMA DE EXECUÇÃO**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Semanas** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** |
| **Idealização do Projeto** | **X** | **X** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Pesquisa de projetos relacionados** |  |  | **X** | **X** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Busca por Materiais** |  |  | **X** | **X** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Elaboração do código** |  |  |  |  | **X** | **X** | **X** |  |  |  |  |  |  |  |  |  |
| **Testes no TinkerCad** |  |  |  |  | **X** | **X** | **X** | **X** |  |  |  |  |  |  |  |  |
| **Apresentação** |  |  |  |  |  |  | **X** | **X** |  |  |  |  |  |  |  |  |
| **Montagem Do Arduino** |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** | **X** | **X** |  |  |
| **Desenvolvimento do App** |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** | **X** |  |  |
| **Teste do App** |  |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** |  |  |
| **Revisão final do projeto** |  |  |  |  |  |  |  |  |  |  |  |  | **X** | **X** | **X** |  |
| **Apresentação Final** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **X** |

**EXPECTATIVA DO PROJETO:**

****

****

**APENDICE**

**CODIGO EM C++ NO APLICATIVO TINKERCAD**

#include <Arduino.h>

#include<DHT.h>

#include<LiquidCrystal\_I2C.h>

#include<string.h>

#include "BluetoothSerial.h"

#include <bluefairy.h>

bluefairy::Scheduler scheduler;

#if !defined(CONFIG\_BT\_ENABLED) || !defined(CONFIG\_BLUEDROID\_ENABLED)

#error Bluetooth is not enabled! Please run `make menuconfig` to and enable it

#endif

// put function declarations

class Display{

private:

    byte endereco;

    LiquidCrystal\_I2C lcd;

public:

  Display(byte posMemoria, int qntdColunas, int qntdLinhas): lcd(posMemoria,qntdColunas,qntdLinhas) {};

  ~Display(){}

  void iniciar(){

    lcd.init();

    this->backlight();

  }

  void clear(){

    lcd.clear();

  }

  void backlight(){

    lcd.backlight();

  }

  void setCursor(int posColuna, int posLinha){

    lcd.setCursor(posColuna, posLinha);

  }

  void printar(String texto, int col, int linha){

    this->setCursor(col,linha);

    lcd.print(texto);

  }

  void printarTemp(float temp, int coluna, int linha){

    this->setCursor(coluna,linha);

    lcd.printf("Atual: %.1f\*C", temp);

  }

};

class LeitorTemperatura{

  private:

    unsigned char indice;

    float leituras[5];

    DHT leitor;

    BluetoothSerial SerialBT;

  public:

    LeitorTemperatura(int pino, BluetoothSerial a): leitor(pino,DHT22), SerialBT(a), indice(0) {};

    ~LeitorTemperatura(){}

    void inicializar(){

      leitor.begin();

    }

    bool pronto() {

      return this->indice == 0;

    }

    void ler() {

      this->leituras[this->indice] = leitor.readTemperature();

      this->indice = (this->indice + 1) % 5;

    }

    float calc() {

      float media = 0;

      for(int i=0;i<5;i++) {

        media += this->leituras[i];

      }

      return media / 5.;

    }

};

Display tela(0x27,16,2);

BluetoothSerial SerialBT;

LeitorTemperatura leitor(2,SerialBT);

char func;

void setup() {

  // put your setup code here, to run once:

  SerialBT.begin("ESP32test"); //Bluetooth device name

  tela.iniciar();

  leitor.inicializar();

  Serial.begin(115200);

  delay(2000);

  tela.clear();

  tela.printar("Iniciando...",0,0);

  scheduler.every(2000, []() {

    leitor.ler();

    if (leitor.pronto()) {

      float media = leitor.calc();

      tela.clear();

      tela.printarTemp(media,0,0);

      SerialBT.write(media);

    }

  });

  scheduler.every(10, []() {

    if (SerialBT.available()) {

          func = (char) SerialBT.read();

          if(func=='1'){

            Serial.write("aumentar");

          }else if(func == '2'){

            Serial.write("diminuir");

          }

    }

  });

}

void loop() {

  scheduler.loop();

}

// put function definitions here:

**Explicação de cada tributo para pontuar:**

-**Codificou classes:** class LeitorTemperatura (lê e exibe a temperatura registrada;

class Display (exibe informações de dados coletados pelo sensor de temperatura).

**-Codificou atributos**: Foram feitos diversos atributos dentro da classe de temperatura por exemplo: byte endereço e float leituras.

**-Codificou métodos:** Foram realizados vários métodos na classe de LeitorTemperatura como ler e calc.

-**Codificou atributos estáticos:** Foram feitos atributos estáticos para se referirem aos componentes, como o display LCD e DHT

**-Codificou Métodos estáticos:** Método calc seria um exemplo de método estático, já que foi feito com o objetivo de permitir que seja invocado em qualquer momento no código.

**-Codificou classes de comportamento:** class Display é focada em usar seus métodos para interagir com display LCD

**-Codificou métodos construtores:** class Display e class LeitorTemperatura possuem construtores com o objetivo de inicializar os atributos principais de cada classe.

**-Código limpo:** cada função é dedicada a realizar apenas uma tarefa.

**-Codificou atributos protegidos e/ou privados:** Foram feitos atributos privados com o objetivo de garantir que não seria possível se relacionar diretamente com os componentes, evitando assim o uso sem seguir as recomendações esclarecidas na documentação do devido componente, Exemplo de atributo privado: class LeitorTemperatura possui o atributo DHT leitor como privado, evitando assim que sejam passada instruções diretamente ao componente fora dos métodos definidos na classe.

**-Instanciou objetos:** Foi criado a classe LeitorTemperatura para registrar as informações e a classe Display para exibilas.

**-Instalou e usou bibliotecas de terceiros:** Sim, utilizamos a biblioteca LiquidCrystal\_I2C.h, responsável pelos comandos de controle de um display LCD I2C.

-**Usou objetos imutáveis:** Temos como exemplo class Display, que após ser criada não pode ser modificada, porém permite que seus métodos controlem o LCD.

**-Usou diagramas UML para discutir a solução:** Sim, achamos muito didático fazer isso devido a facilidade para entendimento de todos, gostamos muito de como conseguimos programas o código e interagir em como o diagrama de manifesta e se liga.

@startuml

class temperatura {

-unit: celsius

float: valor

}

class displayLCD {

+escrever(string)

-cor: azul

}

note right

informa a tempertaura ambiente

end note

temperatura <.. sensor

sensor <|-- displayLCD

note right

coleta dados ambiente

end note

@enduml

**-Codificou classes imutáveis:** Temos como exemplo class temperatura {

-unit: celsius

float: valor

} que permite que usuário não escolha outras temperaturas, a não ser o Celsius.

**-Publicou todo projeto integrador no Gitlab, Github, ou semelhantes:** publicamos e anexamos o link na prova.