1. Descrição do Problema

Sistema embarcado que coleta periodicamente temperatura e umidade via protocolo I2C e envia os dados para o computador por UART/USB. O software no PC registra os dados em arquivo CSV.

Observações
O protocolo I2C não foi utilizado, pois foi confirmado que não era estritamente necessário para a comunicação com o sensor.

2. Requisitos Funcionais

- RF01 - O sistema deve realizar a leitura da temperatura do ambiente - Essencial - Baixa dificuldade.
- RF02 - O sistema deve realizar a leitura da umidade do ambiente - Essencial - Baixa dificuldade.
- RF03 - O sistema deve enviar os dados para um computador - Essencial - Baixa dificuldade.
- RF04 - O sistema deve registrar os dados em um arquivo CSV - Essencial - Baixa dificuldade.

3. Requisitos Não Funcionais

- RNF01 - A leitura das informações do ambiente deve ser realizada em intervalos de, no máximo, 3 segundos, com possibilidade de tempo menor.
- RNF02 - A comunicação serial (UART/USB) entre o sistema embarcado e o computador deve ser estável e com uma taxa de transmissão mínima de 9600 baud (ou superior) para garantir a entrega rápida dos dados.
- RNF03 - O software no PC deve ser capaz de iniciar automaticamente o registro dos dados ao detectar a

conexão do dispositivo embarcado (Plug and Play simplificado).
- RNF06 - O software de registro no PC deve ser executável em sistemas operacionais Windows, Linux e macOS (portabilidade).
- RNF07 - O arquivo CSV gerado deve ter um formato padronizado (Timestamp;Temperatura;Umidade) e ser facilmente legível por ferramentas externas (ex: Excel, Google Sheets).
- RNF08 - O consumo de energia do sistema embarcado deve ser otimizado para permitir operação prolongada, caso seja alimentado por bateria.
- RNF09 - O sistema embarcado deve ser resiliente a falhas temporárias de comunicação (ex: se o computador for desconectado e reconectado, o sistema deve retomar o envio dos dados).

4. Proposta de Solução

Dada a simplicidade do projeto, foram agregados alguns requisitos adicionais e que foram entregues:

- RF04 - O sistema deve se comunicar por rede.
- RF05 - O sistema deve apresentar gráficos para facilitar a visualização dos dados adquiridos no projeto.
- RF06 - O sistema deve ter um simulador acessório de partículas, como proposta de exemplificar que a temperatura é o grau de agitação das moléculas.
- RF07 - O sistema de leitura de temperatura deve ser feito sem sensores prontos.

5. Lista de Materiais Utilizados

- ESP32 DEVKITV1;
- 2 RESISTORES: 1 de 5kΩ e 1 de 10kΩ;
- 1 Termistor NTC 10KΩ;
- Sensor DHT11;
- LCD 16x2;
- Bateria de 9V.

6. Milestones

1. Validação do uso do termistor como sensor de temperatura: Nos reunimos no laboratório com o termistor, um resistor de 10kΩ e o ESP32 para validar a ideia. Como ainda não tínhamos o sensor DHT11 em mãos, utilizamos um sensor de gás MQ-6 para validar a comunicação de múltiplos dispositivos na mesma porta do ESP32, realizando a leitura no termistor. O teste foi um sucesso.
2. Experimento de conexão via WebSocket com o computador e entendimento da dificuldade de implementação: O teste também funcionou como esperado e os dados de tensão do termistor foram enviados via web.
3. Experimento com display LCD para exibição das informações: Com o DHT já disponível, exibimos no LCD a tensão no divisor de tensão do termistor e as leituras de temperatura e umidade. Também foi utilizado um script em Python para gerar o arquivo CSV a partir da leitura dos sensores.
4. Finalização do projeto: Design do hardware definido, software do ESP32 concluído, e interface web com um simulador que mostra “partículas” se movimentando e aumentando sua velocidade de acordo com a intensidade da temperatura.

---

7. Código do Microcontrolador (Firmware Base)

Esta seção detalha a implementação do firmware desenvolvido no PlatformIO, descrevendo a arquitetura de software, bibliotecas e lógica de controle básica.

7.1. Configuração e Bibliotecas

O código configura o ambiente para o hardware ESP32 DOIT DevKit V1 e importa bibliotecas essenciais: `WebSockets` (comunicação), `ArduinoJson` (dados), `DHT` (sensor) e `LiquidCrystal_I2C` (display).

```
src > main.cpp > setup()
#include "DHT.h"
#include <WiFi.h>
#include <Arduino.h>
#include <ArduinoJson.h>
#include <WebSocketsServer.h>
#include <LiquidCrystal_I2C.h>

#define PIN_TERMISTOR 36
float getResistencia(int pin, float voltageUc, float adcResolutionUc, float resistenciaEmSerie);
float readTemperaturaNTC(float resistenciaTermistor, float resistenciaResistorSerie, float voltageUc, float beta);
float calcularCoeficienteBetaTermistor();

/*----- Definição do pino e tipo do sensor DHT -----*/
#define DHT_PIN 15
#define DHT_TYPE DHT11

DHT dht(DHT_PIN, DHT_TYPE);

/*----- Definindo endereço do LCD -----*/
LiquidCrystal_I2C lcd(0x27,16,2);

/*----- ssid/nome, password/senha da rede WiFi -----*/
const char* ssid = "giba-esp32";
const char* password = "esp32teste";

WebSocketsServer webSocket = WebSocketsServer(80); //Porta para hospedagem do servidor ws
```

7.2. Setup e Loop Principal

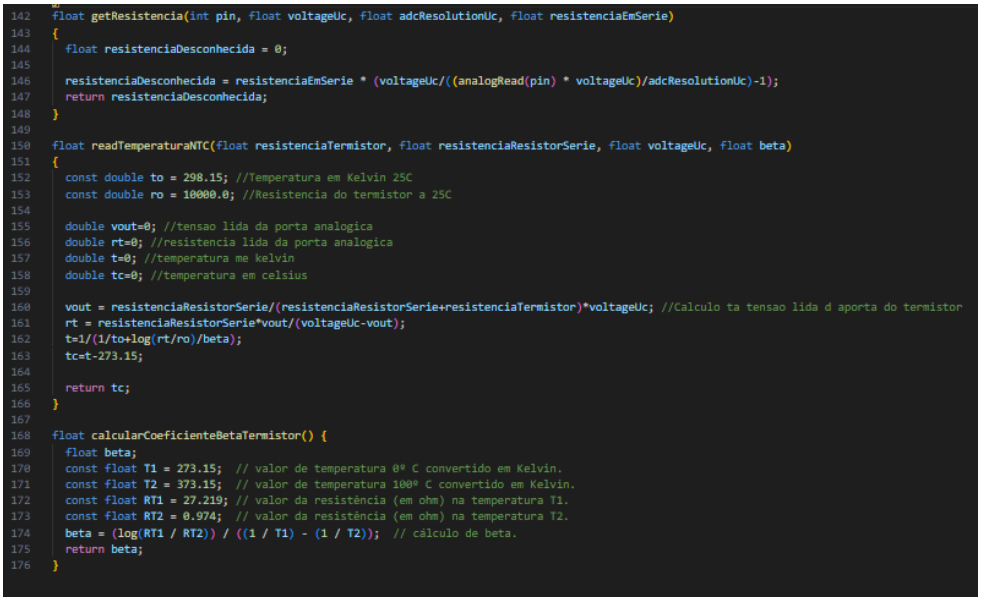
A inicialização do sistema e o ciclo contínuo de leitura e processamento.

```
void setup() {
  Serial.begin(115200);

  Serial.println("-----------------------------------");

  Serial.println("Connecting");
  WiFi.begin(ssid, password);
  while(WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  Serial.println("Connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  webSocket.begin();
  webSocket.onEvent(onWebSocketEvent);
  Serial.println("-----WebSocket Server started!-----");


  pinMode(PIN_TERMISTOR, INPUT);
  dht.begin(); //Inicializando o sensor DHT11

  lcd.init(); //Inicializando a tela LCD
  lcd.backlight();
}
```

```
void loop() {
  webSocket.loop();

  JsonDocument sensors_Document;
  String sensors_Json;

  /*----- coletando dados dos sensores -----*/
  float humidity = dht.readHumidity();
  float temperature_Celsius = dht.readTemperature();
  int termistorValue = analogRead(PIN_TERMISTOR);

  /*----- Convertendo tensão do termisto em temperatura -----*/
  float beta = calcularCoeficienteBetaTermistor();
  float voltageUC = 3.3;
  float adc_Resolution = 4095;
  float known_resistence = 10000.14;
  float resistencia_Termistor = getResistencia(PIN_TERMISTOR, voltageUC, adc_Resolution, known_resistence);
  float temperatura_Termistor = readTemperaturaNTC(resistencia_Termistor, known_resistence, voltageUC, beta);

  //Serial.printf("Tensão do termistor %d\ttemperatura convertida: %f\n", termistorValue, temperatura_Termistor);
  /*----- Imprimindo valores obtidos no LCD -----*/
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.printf("DHT: %.2f%c %.1f", temperature_Celsius, char(223), humidity);
  lcd.setCursor(0,1);
  lcd.printf("Termistor = %d", termistorValue);

  /*----- Verificando leitura do sensor DHT11 e serializando os dados no json -----*/
  if(isnan(humidity) || isnan(temperature_Celsius))
  {
    sensors_Document["humidity"] = 0;
    sensors_Document["temperature"] = 0;
  } else {
    sensors_Document["humidity"] = humidity;
    sensors_Document["temperature"] = temperature_Celsius;
  }
  sensors_Document["termistor"] = termistorValue;
  sensors_Document["termistor_temperature"] = temperatura_Termistor;
  sensors_Document["resistence"] = resistencia_Termistor;

  serializeJson(sensors_Document, sensors_Json);

  //Debugando sensores
  //Serial.printf("Umidade: %f\tTemperatura: %f\tTermistor: %d\n", humidity, temperature_Celsius, termistorValue);
  //Serial.println(sensors_Json);
  //Serial.printf("%.2f;%.2f\n", temperature_Celsius, humidity);

  webSocket.broadcastTXT(sensors_Json); //Transmitidon json com os dados via websocket
  Serial.printf("%.2f;%.2f\n", temperature_Celsius, humidity);
  delay(500);


}
```

7.3. Algoritmos de Conversão Analógica

Funções matemáticas que convertem a resistência elétrica do termistor NTC em temperatura Celsius utilizando o parâmetro Beta.



```
float getResistencia(int pin, float voltageUc, float adcResolutionUc, float resistenciaEmSerie)
{
  float resistenciaDesconhecida = 0;

  resistenciaDesconhecida = resistenciaEmSerie * (voltageUc/((analogRead(pin) * voltageUc)/adcResolutionUc)-1);
  return resistenciaDesconhecida;
}

float readTemperaturaNTC(float resistenciaTermistor, float resistenciaResistorSerie, float voltageUc, float beta)
{
  const double to = 298.15; //Temperatura em Kelvin 25C
  const double ro = 10000.0; //Resistencia do termistor a 25C

  double vout=0; //tensao lida da porta analogica
  double rt=0; //resistencia lida da porta analogica
  double t=0; //temperatura me kelvin
  double tc=0; //temperatura em celsius

  vout = resistenciaResistorSerie/(resistenciaResistorSerie+resistenciaTermistor)*voltageUc; //Calculo ta tensao lida d aporta do termistor
  rt = resistenciaResistorSerie*vout/(voltageUc-vout);
  t=1/(1/to+log(rt/ro)/beta);
  tc=t-273.15;

  return tc;
}

float calcularCoeficienteBetaTermistor() {
  float beta;
  const float T1 = 273.15;  // valor de temperatura 0º C convertido em Kelvin.
  const float T2 = 373.15;  // valor de temperatura 100º C convertido em Kelvin.
  const float RT1 = 27.219; // valor da resistência (em ohm) na temperatura T1.
  const float RT2 = 0.974;  // valor da resistência (em ohm) na temperatura T2.
  beta = (log(RT1 / RT2)) / ((1 / T1) - (1 / T2));  // cálculo de beta.
  return beta;
}
```

---

8. Código Python que Implementa CSV

Este script é executado no computador (Host) e cumpre o requisito de portabilidade (RNF06). Ele escuta a porta serial conectada ao ESP32, captura a string de dados, adiciona o timestamp do sistema e salva em um arquivo `dados.csv`.

Principais funcionalidades:

- Detecção automática da porta (opcional);
- Parsing da string recebida;
- Escrita em disco em tempo real.

```
/*----- Função do comportamento em retorno do servidor -----*/
void onWebSocketEvent(uint8_t num, WStype_t type, uint8_t* payload, size_t length)
{
  switch(type)
  {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] Disconnected!\n", num);
      break;

    case WStype_CONNECTED:{
      IPAddress ip = webSocket.remoteIP(num);
      Serial.printf("[%u] Connection from ", num);
      Serial.println(ip.toString());
    }
      break;

    case WStype_TEXT:
      Serial.printf("[%u] Text: %s\n", num, payload);
      webSocket.sendTXT(num, payload);
      break;

    case WStype_BIN:
    case WStype_ERROR:
    case WStype_FRAGMENT:
    case WStype_FRAGMENT_TEXT_START:
    case WStype_FRAGMENT_BIN_START:
    case WStype_FRAGMENT_FIN:
    default:
      break;
  }
}
```

9. Servidor Web (WebSocket)

Esta seção cobre a implementação do RF04 (Comunicação por rede). O ESP32 atua como um servidor WebSocket assíncrono na porta 80.

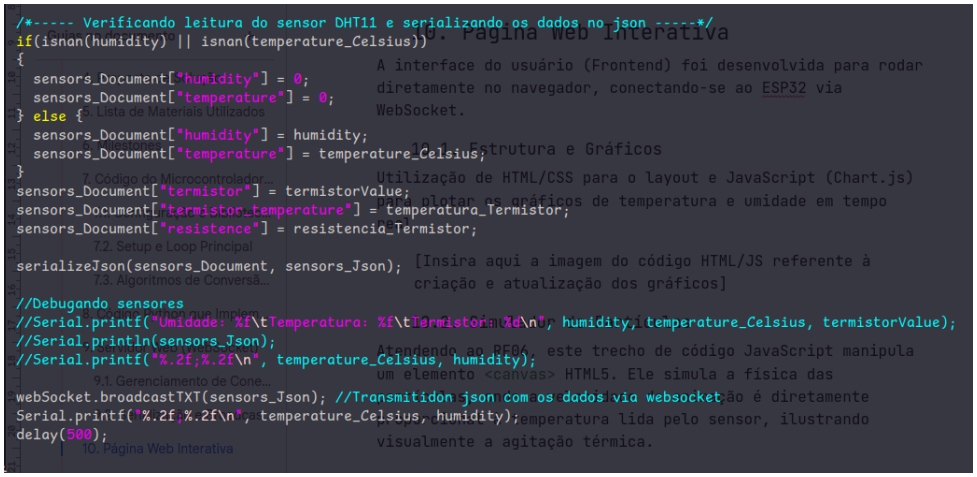
9.1. Gerenciamento de Conexões

```
# ---------------- LER SERIAL ----------------
def ler_serial():
    global rodando

    if not conectar_serial():
        return

    escrever_cabecalho = not os.path.exists(arquivo_csv) or os.stat(arquivo_csv).st_size == 0

    with open(arquivo_csv, "a", newline="") as arquivo:
        writer = csv.writer(arquivo, delimiter=";")

        if escrever_cabecalho:
            writer.writerow(["Data", "Temperatura", "Umidade"])

        while rodando:
            try:
                linha = ser.readline().decode(errors="ignore").strip()
                dados = linha.split(";")

                if len(dados) == 2:
                    temperatura, umidade = dados
                    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

                    writer.writerow([timestamp, temperatura, umidade])
                    print(timestamp, temperatura, umidade)

            except:
                break

    status_label.config(text="Execução parada", fg="red")
```

O código abaixo mostra a função de *callback* que gerencia novos clientes conectados, desconexões e o recebimento de mensagens de controle.

9.2. Serialização e Broadcast JSON

Para enviar os dados para a interface web, o sistema cria um objeto JSON contendo todas as leituras dos sensores e faz o *broadcast* (envio simultâneo) para todos os clientes conectados.



```
/*----- Verificando leitura do sensor DHT11 e serializando os dados no json -----*/
if(isnan(humidity) || isnan(temperature_Celsius))
{
  sensors_Document["humidity"] = 0;
  sensors_Document["temperature"] = 0;
} else {
  sensors_Document["humidity"] = humidity;
  sensors_Document["temperature"] = temperature_Celsius;
}
sensors_Document["termistor"] = termistorValue;
sensors_Document["termistor_temperature"] = temperatura_Termistor;
sensors_Document["resistence"] = resistencia_Termistor;

serializeJson(sensors_Document, sensors_Json);

//Debugando sensores
//Serial.printf("Umidade: %f\tTemperatura: %f\tTermistor: %d\n", humidity, temperature_Celsius, termistorValue);
//Serial.println(sensors_Json);
//Serial.printf("%.2f;%.2f\n", temperature_Celsius, humidity);
webSocket.broadcastTXT(sensors_Json); //Transmitindo json com os dados via websocket
Serial.printf("%.2f;%.2f\n", temperature_Celsius, humidity);
delay(500);
```

---

10. Página Web Interativa

A interface do usuário (Frontend) foi desenvolvida para rodar diretamente no navegador, conectando-se ao ESP32 via WebSocket.

10.1. Estrutura e Gráficos

Utilização de HTML/CSS para o layout e JavaScript (Chart.js) para plotar os gráficos de temperatura e umidade em tempo real.

```
<!-- Gráficos -->
<div class="charts-grid">
  <div class="chart-container">
    <div class="chart-title">Temperatura do Termistor x Tempo</div>
    <div class="chart-wrapper">
      <canvas id="termistorTempChart"></canvas>
    </div>
  </div>
  <div class="chart-container">
    <div class="chart-title">Termistor ADC x Tempo</div>
    <div class="chart-wrapper">
      <canvas id="termistorChart"></canvas>
    </div>
  </div>
  <div class="chart-container">
    <div class="chart-title">Resistência x Tempo</div>
    <div class="chart-wrapper">
      <canvas id="resistanceChart"></canvas>
    </div>
  </div>
```

10.2. Simulador de Partículas

Atendendo ao RF06, este trecho de código JavaScript manipula um elemento <canvas> HTML5. Ele simula a física das partículas, onde a velocidade de animação é proporcional à temperatura lida pelo sensor, ilustrando visualmente a agitação térmica.

10.3. Transformar tensão em temperatura do termistor:

Para fazer o sensor é bem simples você vai precisar desses materiais:

- ESP32
- Resistor
- Termistor (um resistor que altera sua resistêsia de acordo com a temperatura)

A estratégia de transformar medida elétricas em temperatra baseia-se em que Stein-Hart desenvolveu um modelo matemático que relaciona resistência elétrica e temperatura, a equação descreve uma relação logaritma entre a temperatura e resistencia em semicondutores como o termistor. A equação é dada por:

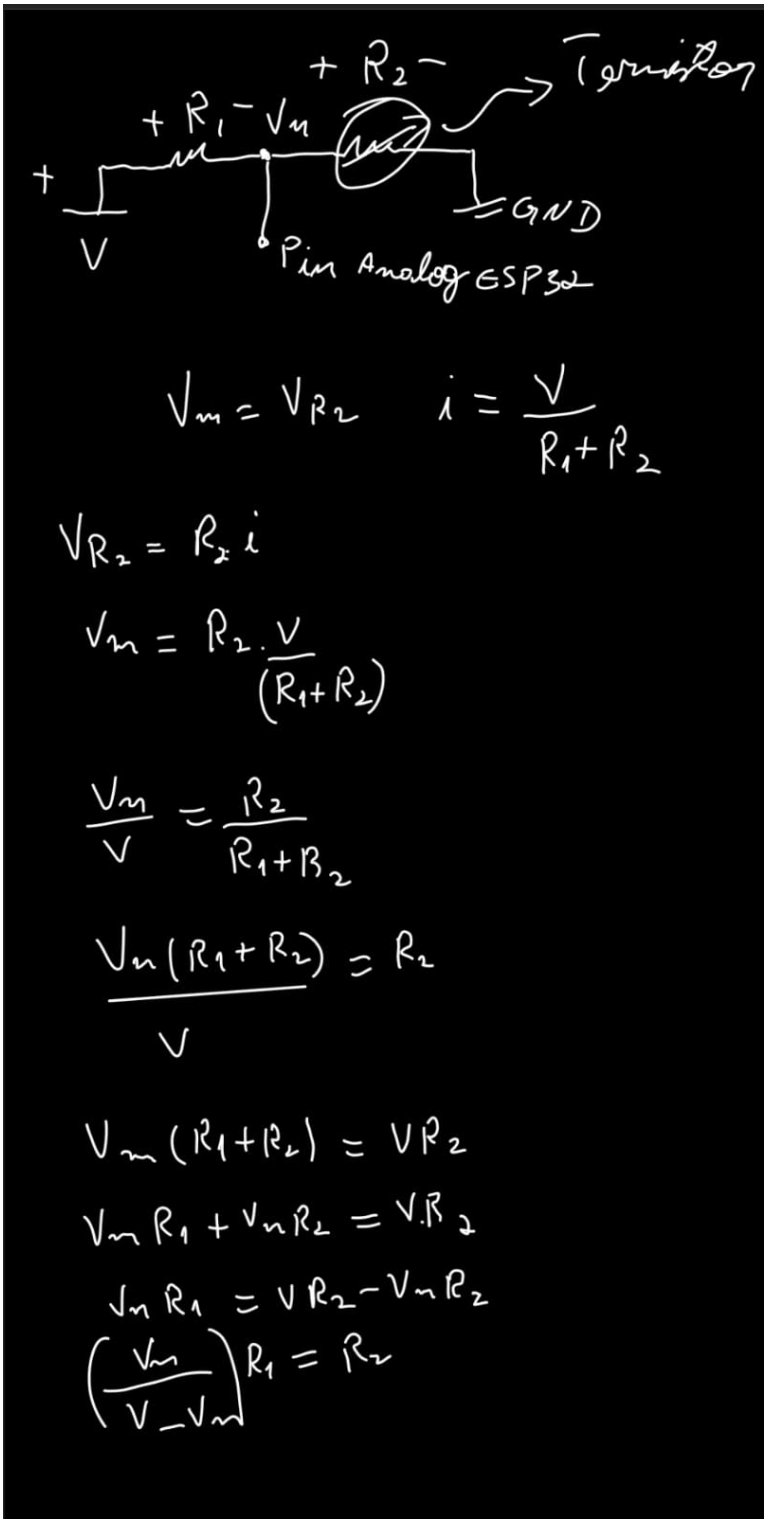
$$\frac{1}{T} = A + B\ln R + C(\ln R)^3,$$

A, B e C são constantes, elas podem ser encontradas experimentalmente, T é a temperatura em Kelvin e R é a resistencia no semicondutor.

O esp32 é capaz de ler tensão atravez dos pinos analógicos de 12bits para transformar a leitura em tensão basta multiplicar o valor de leitura pela tensão de saída dividido pelo valor máximo de leitura da porta do esp que é de 12bits logo 2^12 ou seja 4096:

v = pin_medido*(Vout/4096)

Como a fórmula de Stein-Hart relaciona resistência e não tensão fazemos um divisor de tensão, aplicamos uma tensão conhecida e coolocamos uma resistência conhecida para calcular-mos a função que relaciona tensão e resistência no termistor:



$$V_m = V_{R2} \qquad i = \frac{V}{R_1 + R_2}$$

$$V_{R_2} = R_2 \, i$$

$$V_m = R_2 \cdot \frac{V}{(R_1 + R_2)}$$

$$\frac{V_m}{V} = \frac{R_2}{R_1 + R_2}$$

$$\frac{V_m (R_1 + R_2)}{V} = R_2$$

$$V_m (R_1 + R_2) = V R_2$$

$$V_m R_1 + V_m R_2 = V R_2$$

$$V_m R_1 = V R_2 - V_m R_2$$

$$\left(\frac{V_m}{V - V_m}\right) R_1 = R_2$$

10.4. Webassembly canvas do simulador de partículas:

Já é sabido que o navegador é uma máquina virtual e o que o webassembly é compilar uma outra linguagem para o código de máquina dessa linguagem, o que permite rodar códigos complexos com maior performance e expandir os horizontes da web.

Nesse projeto foi utilizado o C com a biblioteca raylib para fazer o simulador de partículas, e o compilador Emscripten.

O javascript lida com as requisições e o C cuida de toda lógica de execução das partículas.