PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

ESCOLA POLITÉCNICA

ENGENHARIA DE SOFTWARE

Aquário Automatizado

Curitiba

2025

MATHEUS MARCONDES MULLER

GERMANO LAGANA

VINICIUS PADILHA

Aquário Automatizado

Trabalho apresentado na disciplina de Performance em Sistemas Ciberfísicos, do curso de Engenharia de Software da Pontifícia Universidade Católica do Paraná, sob orientação do professor Fabio Bettio.

Curitiba

2025

SUMÁRIO

1 INTRODUÇÃO

O projeto consiste no desenvolvimento de um aquário automatizado, integrando o microcontrolador ESP32 para controlar a alimentação dos peixes, monitorar a temperatura da água e regular a iluminação de forma inteligente. O sistema também enviará dados em tempo real para o usuário, permitindo o acompanhamento remoto das condições do aquário por meio de um aplicativo ou plataforma web.

2 JUSTIFICATIVA

Os sistemas ciberfísicos (CPS) combinam componentes físicos e computacionais, permitindo a interação em tempo real entre hardware, software e o ambiente. Este projeto se encaixa perfeitamente nesse conceito, pois no projeto existem:

- Integração entre Hardware e Software.
- Monitoramento em Tempo Real.
- Autonomia e Controle Remoto.
- Aplicação em IoT (Internet das Coisas).

3 TECNOLOGIAS UTILIZADAS NO AQUÁRIO

Componentes:

- ESP32
- Motor de Passo 28BYJ-48 + Driver ULN2003
- Módulo RTC DS3231
- LCD 16x2 com Módulo I2C
- Micro-USB/Fonte
- Resistor 150Ω ou 1kΩ
- LED 3mm
- Interruptor de Dois Polos

Bibliotecas:

- RTClib (da Adafruit)

- LiquidCrystal_I2C
- Stepper

Protocolo de Comunicação:

- I2C, HTTP/HTTPS

Medidor de Temperatura:

Componentes:

- DS18B20 (à prova d'água)
- DHT22
- NTC Thermistor

Bibliotecas:

- Wire.h
- LiquidCrystal_I2C.h
- OneWire.h e DallasTemperature.h

Protocolo de Comunicação:

- 1-Wire

Luz Automática - Componentes:

- Relé ou MOSFET

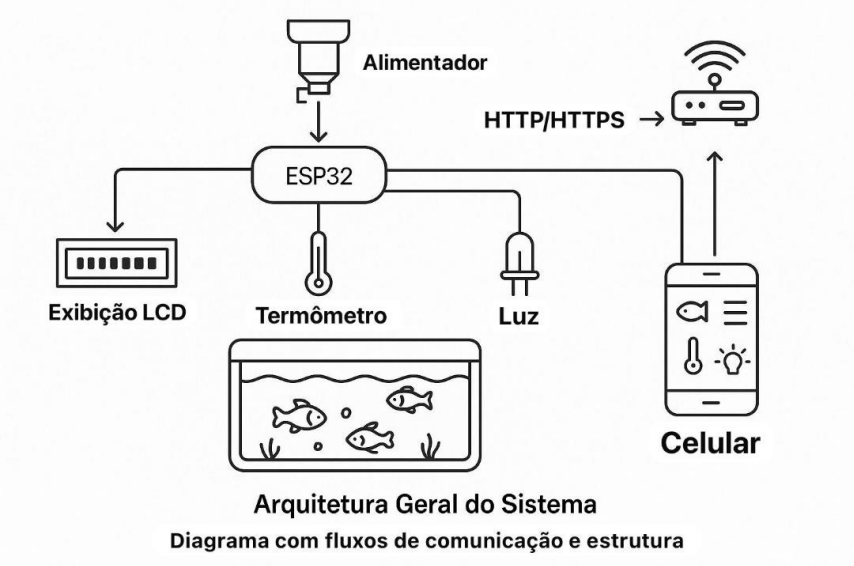
Bibliotecas:

- Wire.h
- RTClib.h

Protocolo de Comunicação:

- I2C

4 ARQUITETURA GERAL DO SISTEMA



5 CRONOGRAMA DE EXECUÇÃO

1 – Pesquisa e Definição: Fechar lista de componentes, bibliotecas, estrutura geral.

2 – Montagem Inicial: Montar ESP32 com RTC, LCD e motor de passo (Sem auário real ainda)

3 – Programação Básica: Codificar RTC, controle do motor, e controle de luz (em testes locais)

4 – Integração dos Sensores: Integrar sensores de temperatura e leitura no ESP32.

5 – Desenvolvimento de Comunicação: Enviar dados via HTTP/HTTPS para uma plataforma simples (Ex: ThingSpeak, Blynk, ou Firebase).

6 – Desenvolvimento da Interface: Criar aplicativo mobile ou página web para visualização dos dados.

7 – Testes e ajustes: Testes de tempo, alimentação, temperatura e luz automatizada. Correções.

8 – Instalação no aquário: Instalar fisicamente no aquário real, com protetor para eletrônicos.

9 – Testes reais: Monitorar funcionamento contínuo no aquário.

10 – Entrega final: Finalizar documentação e preparar apresentação/projeto final.

6 TESTES

Foram realizados testes nos principais módulos:

- Teste do Alimentador de Peixes

```
/* ******************** Alimentador para peixes do Manual do Mundo (Adaptado ESP32) ********************
   Adaptado por: Matheus Muller, Vinicius Padilha, Germano Lagana
   Rev.: 02
   Data: 28.04.2025
********************************************************************************** */

// Inclusão das bibliotecas
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Stepper.h>
#include <RTClib.h>

RTC_DS3231 rtc;

// Define o endereço do LCD e o tamanho da tela
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Define as entradas onde o motor de passo está conectado
#define in1 13
#define in2 12
#define in3 14
#define in4 15

int passosPorAcionamento = 32;

int passosRefeicao = 4;

// Define o motor de passo
Stepper mp(passosPorAcionamento, in1, in3, in2, in4);

// Parâmetros de horário que serão atualizados
int horaAtual, minutoAtual;
// Parâmetros horários de alimentação
int demosComida1, demosComida2;

/* ******************** CONFIGURAÇÃO DE HORÁRIO PARA ALIMENTAÇÃO ********************/

// Primeira alimentação
#define horaAlimentacao1 20
#define minutoAlimentacao1 30

// Segunda alimentação
#define horaAlimentacao2 8
#define minutoAlimentacao2 30

/********************************************************************************** */

void setup() {
  // Inicializa a comunicação serial (opcional para depuração)
  Serial.begin(115200);

  // Inicializa o LCD
  lcd.init();
  lcd.backlight();

  // Inicializa o motor
  mp.setSpeed(500);

  // Inicializa o RTC
  if (!rtc.begin()) {
    Serial.println("Não foi possível encontrar o módulo RTC");
    while (1);
  }

  // Se o RTC perdeu a alimentação, ajustar o horário:
  if (rtc.lostPower()) {
    Serial.println("RTC perdeu a alimentação, ajustando hora...");
    // Define o horário inicial (ano, mês, dia, hora, minuto, segundo)
    rtc.adjust(DateTime(2025, 4, 28, 12, 0, 0));
```

```
    rtc.adjust(DateTime(2025, 4, 28, 12, 0, 0));
  }

  demosComida1 = 0;
  demosComida2 = 0;
}

void loop() {
  // Obtém a hora atual
  DateTime now = rtc.now();
  horaAtual = now.hour();
  minutoAtual = now.minute();

  // Verifica se é o horário da primeira alimentação
  if (horaAtual == horaAlimentacao1 && minutoAtual == minutoAlimentacao1 && demosComida1 == 0) {
    alimentarPeixe();
    demosComida1 = 1;
  }

  // Verifica se é o horário da segunda alimentação
  if (horaAtual == horaAlimentacao2 && minutoAtual == minutoAlimentacao2 && demosComida2 == 0) {
    alimentarPeixe();
    demosComida2 = 1;
  }

  // Atualiza LCD
  if (demosComida1 == 0 || (demosComida1 == 1 && demosComida2 == 1)) {
    horarioNaTela(now);
    alimentacao1Tela();
  }

  if (demosComida1 == 1 && demosComida2 == 0) {
    horarioNaTela(now);
    alimentacao2Tela();
  }

  // Reseta alimentação à meia-noite
  if (horaAtual == 0 && minutoAtual == 0) {
    demosComida1 = 0;
    demosComida2 = 0;
  }

  delay(1000);
}

// Função que alimenta os peixes
void alimentarPeixe() {
  for (int i = 0; i < passosRefeicao; i++) {
    mp.step(passosPorAcionamento);
  }
  // Desliga bobinas do motor
  digitalWrite(in1, LOW);
  digitalWrite(in2, LOW);
  digitalWrite(in3, LOW);
  digitalWrite(in4, LOW);
  delay(1000);
}

// Função que exibe o horário atual no LCD
void horarioNaTela(DateTime now) {
  char horaRelogioStr[17];
  sprintf(horaRelogioStr, "Hora: %02d:%02d:%02d", now.hour(), now.minute(), now.second());
  lcd.setCursor(0, 0);
  lcd.print(horaRelogioStr);
}

// Função que exibe o próximo horário de alimentação 1
void alimentacao1Tela() {
  char horaMinutoStr[17];
```

```
// Função que exibe o próximo horário de alimentação 1
void alimentacao1Tela() {
  char horaMinutoStr[17];
  sprintf(horaMinutoStr, "Prox: %02d:%02d:00", horaAlimentacao1, minutoAlimentacao1);
  lcd.setCursor(0, 1);
  lcd.print(horaMinutoStr);
}

// Função que exibe o próximo horário de alimentação 2
void alimentacao2Tela() {
  char horaMinutoStr[17];
  sprintf(horaMinutoStr, "Prox: %02d:%02d:00", horaAlimentacao2, minutoAlimentacao2);
  lcd.setCursor(0, 1);
  lcd.print(horaMinutoStr);
}
```

- Teste do Bot Telegram

```ino
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <Wire.h>
#include <RTClib.h>

// WiFi
const char* ssid = "SEU_SSID";
const char* password = "SUA_SENHA_WIFI";

// Telegram
#define BOT_TOKEN "SEU_BOT_TOKEN"  // Bot token fornecido pelo @BotFather
#define CHAT_ID "SEU_CHAT_ID"      // Chat ID para onde enviar a mensagem

WiFiClientSecure client;
UniversalTelegramBot bot(BOT_TOKEN, client);

// RTC
RTC_DS3231 rtc;

// Simulação de dados de temperatura e luz
float temperaturaAtual = 25.3; // Valor exemplo
bool luzLigada = false; // Estado da luz

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

  // Conecta no WiFi
  WiFi.begin(ssid, password);
  Serial.print("Conectando no WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" Conectado!");

  // Permite HTTPS (necessário para o Telegram)
  client.setInsecure();

  // Inicializa o RTC
  if (!rtc.begin()) {
    Serial.println("Erro ao inicializar RTC!");
    while (1);
  }

  // (opcional) ajustar o RTC caso necessário:
  // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  // Aguarda conexão
  delay(1000);
}

void loop() {
    // Lê o horário atual
    DateTime now = rtc.now();

    // (opcional) Simular leitura real da temperatura
    temperaturaAtual += random(-5, 5) * 0.1;

    // Formatar mensagem
    String mensagem = "📋 Status do Sistema\n";
    mensagem += "🕒 Hora atual: ";
    mensagem += String(now.hour()) + ":" + String(now.minute()) + ":" + String(now.second()) + "\n";
    mensagem += "🌡️ Temperatura: ";
    mensagem += String(temperaturaAtual, 1) + "°C\n";
    mensagem += "💡 Luz: ";
    mensagem += luzLigada ? "Ligada" : "Desligada";

    // Envia para o Telegram
    bot.sendMessage(CHAT_ID, mensagem, "");

    Serial.println("Mensagem enviada!");

    // Aguarda 1 hora para enviar de novo (ou ajuste como quiser)
    delay(3600000);
  }
```

- Teste da Luz Automática

```ino
#include <Wire.h>
#include <RTClib.h>

#define RELAY_PIN 16  // GPIO para controlar a luz (relé)

// Inicializar o RTC
RTC_DS3231 rtc;  // Relógio de tempo real

// Variáveis de horário
int horaAtual, minutoAtual;
int horaLigamentoLuz = 18, minutoLigamentoLuz = 0; // 18:00
int horaDesligamentoLuz = 6, minutoDesligamentoLuz = 0; // 06:00

void setup() {
  // Inicializar o LCD e RTC
  Serial.begin(115200);
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);  // Luz apagada inicialmente

  // Verifica se o RTC está funcionando
  if (!rtc.begin()) {
    Serial.println("Não foi possível encontrar o RTC");
    while (1);
  }

  // Ajuste de tempo (faça isso manualmente se necessário)
  // rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
}

void loop() {
  // Obter hora atual
  DateTime now = rtc.now();
  horaAtual = now.hour();
  minutoAtual = now.minute();

  // Verificar se é hora de ligar a luz
  if (horaAtual == horaLigamentoLuz && minutoAtual == minutoLigamentoLuz) {
    digitalWrite(RELAY_PIN, HIGH);  // Liga a luz
  }

  // Verificar se é hora de desligar a luz
  if (horaAtual == horaDesligamentoLuz && minutoAtual == minutoDesligamentoLuz) {
    digitalWrite(RELAY_PIN, LOW);  // Desliga a luz
  }

  // Atraso de 1 segundo para evitar leituras rápidas demais
  delay(1000);
}
```

- Teste do Medidor de Temperatura

```ino
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>

// Pino para o sensor DS18B20 (GPIO15)
#define ONE_WIRE_BUS 15

// Inicializa a comunicação OneWire e o sensor
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Inicializa o LCD (endereço I2C 0x27)
LiquidCrystal_I2C lcd(0x27, 16, 2); // (endereço, 16 colunas, 2 linhas)

void setup() {
  // Inicia a comunicação serial
  Serial.begin(115200);

  // Inicia o LCD
  lcd.init();
  lcd.backlight();

  // Inicia o sensor DS18B20
  sensors.begin();
}

void loop() {
  // Solicita a leitura da temperatura do sensor
  sensors.requestTemperatures();

  // Obtém a temperatura do primeiro sensor (DS18B20)
  float temperaturaC = sensors.getTempCByIndex(0);

  // Verifica se a leitura foi bem-sucedida
  if (temperaturaC != DEVICE_DISCONNECTED_C) {
    // Exibe a temperatura no LCD
    lcd.clear();  // Limpa o LCD
    lcd.setCursor(0, 0);
    lcd.print("Temperatura:");
    lcd.setCursor(0, 1);
    lcd.print(temperaturaC);
    lcd.print(" C");
  } else {
    // Se o sensor não estiver conectado corretamente
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Erro no sensor");
  }

  // Exibe a temperatura também no monitor serial
  Serial.print("Temperatura: ");
  Serial.print(temperaturaC);
  Serial.println(" C");

  // Atraso de 1 segundo antes de nova leitura
  delay(1000);
}
```

7 CONSIDERAÇÕES FINAIS

O aquário automatizado demonstrou ser uma solução viável e eficiente, integrando recursos de automação e monitoramento remoto para facilitar o cuidado com peixes e promover um ambiente aquático saudável.

8 REFERÊNCIAS

MULLER, M. Aquário Automatizado. Disponível em: https://github.com/MM-Muller/aquarioAutomatizado. Acesso em: 29 abr. 2025.