PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

MATHEUS MARCONDES MULLER
VINICIUS PADILHA
GERMANO LAGANA

PERFORMANCE EM SISTEMAS CIBERFÍSICOS (TURMA 3º A)

Curitiba-PR
2025

MATHEUS MARCONDES MULLER
VINICIUS PADILHA
GERMANO LAGANA

PERFORMANCE EM SISTEMAS CIBERFÍSICOS (TURMA 3º A)

Trabalho de Conclusão de Curso apresentado ao curso de Performance em Sistemas Ciberfísicos, da PUCPR, como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Software.

Orientador: Fabio Garcez Bettio
Coorientador: Fabio Garcez Bettio

Curitiba-PR
2025

RESUMO

Este projeto inovador foca no desenvolvimento de um sistema automatizado para aquários domésticos, utilizando o microcontrolador ESP32 para simplificar e otimizar os cuidados com peixes e o ambiente aquático. A ideia surgiu da necessidade de superar os desafios da supervisão manual, alimentação, controle de temperatura e iluminação, especialmente durante ausências prolongadas. O sistema, que opera como um sistema ciberfísico baseado em IoT, monitora a temperatura da água, automatiza a alimentação dos peixes em horários pré-programados e controla a iluminação com base em sensores e tempo. Ele também envia dados em tempo real, permitindo o controle remoto via Web. Entre as tecnologias utilizadas, destacam-se o ESP32 pela conectividade Wi-Fi e baixo consumo de energia, sensores como DS18B20 para temperatura, um motor de passo para o alimentador e um display LCD para exibir informações. Testes isolados de cada módulo – motor de passo, RTC, display LCD, alimentador e controle de iluminação – confirmaram a eficácia do sistema. Este projeto demonstra com sucesso a aplicação da automação em aquários, oferecendo maior controle, conforto e segurança. Futuramente, poderá ser expandido com mais sensores e conexão à nuvem.

Palavras-chave: Aquário Inteligente; Maker.

ABSTRACT

This innovative project focuses on developing an automated system for domestic aquariums, using the ESP32 microcontroller to simplify and optimize the care of fish and the aquatic environment. The idea arose from the need to overcome the challenges of manual supervision, feeding, temperature control, and lighting, especially during prolonged absences. The system, which operates as an IoT-based cyber-physical system, monitors water temperature, automates fish feeding at pre-programmed times, and controls lighting based on sensors and time. It also sends real-time data, allowing for remote control via the Web. Among the technologies used, the ESP32 stands out for its Wi-Fi connectivity and low power consumption, along with sensors like the DS18B20 for temperature, a stepper motor for the feeder, and an LCD display to show information. Isolated tests of each module – stepper motor, RTC, LCD display, feeder, and lighting control – confirmed the system's effectiveness. This project successfully demonstrates the application of automation in aquariums, offering greater control, comfort, and safety. In the future, it can be expanded with more sensors and cloud connectivity.

Keywords: Smart aquarium; Maker.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 **INTRODUÇÃO**

A crescente busca por conveniência e otimização de tempo no cotidiano moderno tem impulsionado o desenvolvimento de soluções tecnológicas para tarefas que, tradicionalmente, demandavam supervisão e execução manuais. No contexto do cuidado com animais de estimação, especificamente aquários domésticos, a manutenção diária, que envolve alimentação, controle de temperatura e iluminação, pode ser desafiadora, especialmente em períodos de ausência prolongada dos cuidadores. A negligência nessas tarefas compromete diretamente o bem-estar e a qualidade de vida dos animais aquáticos.

Nesse cenário, os sistemas ciberfísicos (CPS) e a Internet das Coisas (IoT) emergem como pilares fundamentais para a criação de ambientes inteligentes e automatizados. Tais sistemas integram hardware, software e conectividade em tempo real, permitindo monitoramento e controle eficientes de processos físicos. Este trabalho propõe o desenvolvimento de um "Aquário Automatizado", um sistema inteligente que visa mitigar as falhas humanas na rotina de manutenção e proporcionar maior controle, conforto e segurança ao ambiente aquático.

O objetivo geral deste projeto é desenvolver um sistema automatizado para aquários utilizando o microcontrolador ESP32 para o monitoramento e controle da alimentação, temperatura e iluminação. Para tanto, foram definidos objetivos específicos que incluem o controle automático da alimentação dos peixes, o monitoramento e exibição em tempo real da temperatura da água, o controle da iluminação baseado em tempo ou sensores, e a implementação de uma interface remota para interação via Web.

A escolha do microcontrolador ESP32 justifica-se pela sua conectividade Wi-Fi integrada, baixo consumo de energia e ampla compatibilidade com diversos sensores e atuadores, características que o tornam superior a outras plataformas para aplicações conectadas. A arquitetura do sistema contempla a integração de sensores (DS18B20, DHT22, NTC), atuadores (motor de passo e fita de LED RGB), display LCD e um módulo RTC DS3231 de alta precisão. A comunicação eficiente entre os componentes é garantida pelo protocolo I2C, otimizando o uso de pinos do microcontrolador.

Este trabalho demonstra a viabilidade e a eficácia da aplicação de sistemas ciberfísicos na automação de aquários, contribuindo para a melhoria da qualidade de vida dos animais aquáticos e a praticidade para seus cuidadores. Para a transparência e replicabilidade do estudo, o código-fonte completo do projeto está publicamente disponível no repositório GitHub, acessível em https://github.com/MM-Muller/aquarioAutomatizado. A seguir, serão detalhados a justificativa, os objetivos, as tecnologias empregadas, a arquitetura do sistema, os testes realizados e as

considerações finais sobre o projeto.

2 **ESPECIFICAÇÃO DO SISTEMA E FUNCIONALIDADES**

O sistema de Aquário Automatizado foi concebido para oferecer uma gestão eficiente e autônoma das condições essenciais para a manutenção de um ambiente aquático saudável em aquários domésticos. As funcionalidades foram desenvolvidas com base nos objetivos de monitoramento, controle e interação remota, utilizando o microcontrolador ESP32 como núcleo principal.

2.1 MONITORAMENTO DO AQUÁRIO

O sistema realiza o monitoramento contínuo de parâmetros cruciais para a vida aquática:

1. Monitoramento de Temperatura da Água: A temperatura da água é monitorada em tempo real por meio de sensores DS18B20. Essa informação é crucial para garantir que a temperatura se mantenha dentro da faixa ideal para as espécies de peixes do aquário, evitando estresses térmicos que podem comprometer a saúde dos animais. Os dados de temperatura são exibidos no display LCD e podem ser acessados remotamente.

2. Exibição de Dados: As informações de temperatura atual, bem como o horário atual e os próximos horários de alimentação, são exibidas em um display LCD 16x2 com módulo I2C. Isso permite uma rápida visualização local das condições do aquário.

2.2 CONTROLE AUTOMATIZADO

As funções de controle são projetadas para gerenciar automaticamente a rotina do aquário:

1. Controle de Alimentação dos Peixes: O sistema é capaz de acionar a alimentação dos peixes em horários pré-programados. Para isso, utiliza um Motor de Passo 28BYJ-48 com Driver ULN2003, que dispensa a ração de forma precisa. Dois horários de alimentação podem ser definidos (ex: 20:30 e 08:30), e o sistema garante que a alimentação ocorra nesses momentos, repondo o status após a meia-noite. Essa funcionalidade é vital para manter a saúde dos peixes, especialmente em ausências prolongadas.

2. Controle de Iluminação: A iluminação do aquário é controlada com base em tempo ou sensores, utilizando uma fita de LED RGB. O sistema permite a programação de horários para ligar (ex: 18:00) e desligar (ex: 06:00) a luz, simulando o ciclo dia/noite natural e promovendo o bem-estar dos peixes e plantas.

2.3 INTERATIVIDADE E CONECTIVIDADE

O sistema oferece recursos de conectividade para interação e controle remoto:

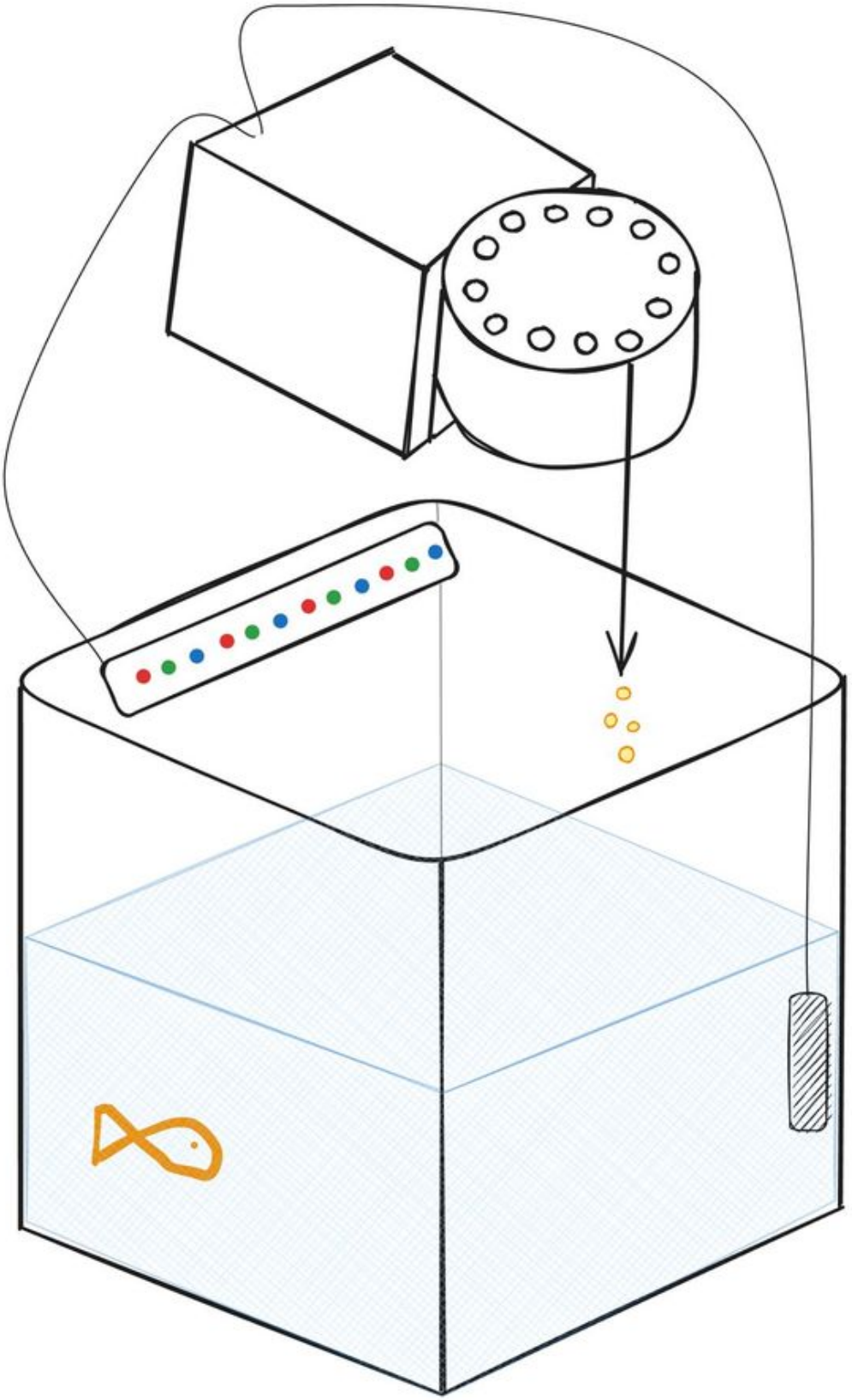
1. Comunicação de Dados em Tempo Real: O projeto envia dados em tempo real por meio de rede (Wi-Fi), possibilitando o controle remoto do aquário. A comunicação é estabelecida via HTTP/HTTPS.

2. Interface Remota (Telegram ou Web): Foi desenvolvida uma interface remota que permite ao usuário interagir com o aquário de forma conveniente, seja via aplicativo Telegram ou uma interface web. Esta funcionalidade possibilita a consulta de informações sobre o estado do sistema (temperatura, status da luz, horário) e o envio de comandos, como ligar/desligar a luz ou verificar o estado do aquário. O uso do Telegram provê uma forma acessível de controle a distância.

3. Relógio de Tempo Real (RTC): A utilização do Módulo RTC DS3231 garante a precisão e a estabilidade na contagem do tempo, fundamental para as funções de alimentação e iluminação programadas, mesmo em caso de perda de energia no microcontrolador.

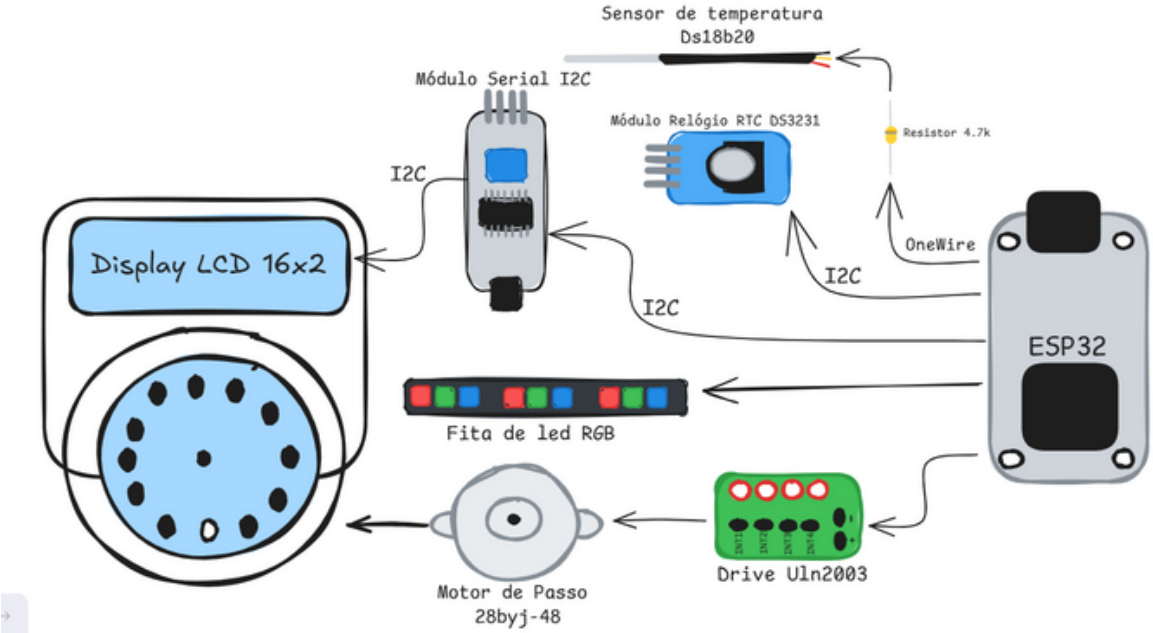
3 **DIAGRAMA DA ARQUITETURA**

Desenho 1 — Arquitetura Geral do projeto do Aquário Automatizado



Fonte: Os autores (2025).

Desenho 2 — Diagrama da Arquitetura Ciberfísica



Fonte: Os autores (2025).

4 **RELATÓRIO DE DESEMPENHO E TESTES**

Esta seção detalha o processo de validação das funcionalidades do sistema "Aquário Automatizado", abordando a metodologia de testes empregada e apresentando os resultados obtidos. O objetivo foi verificar a correta operação de cada módulo isoladamente e a integração entre eles, garantindo que o sistema atenda aos requisitos especificados.

4.1 **METODOLOGIA DE TESTES**

A metodologia de testes adotada seguiu uma abordagem modular e incremental, iniciando-se com testes isolados de cada componente de hardware e suas respectivas funcionalidades de software, e progredindo para testes de integração. Essa estratégia permitiu identificar e corrigir falhas em etapas precoces do desenvolvimento, antes da complexidade de um sistema totalmente integrado.

Os testes foram conduzidos em um ambiente de bancada, simulando as condições de operação de um aquário doméstico. Foram utilizados recursos de monitoramento serial para depuração e verificação dos dados processados pelos microcontroladores, além de observação direta do comportamento dos atuadores (motor de passo, LEDs) e do display LCD.

4.2 TESTES ISOLADOS

Foram realizados testes específicos para cada componente principal do sistema, conforme detalhado a seguir:

4.2.1 **Teste do Motor de Passo (28BYJ-48)**

Este teste teve como objetivo verificar o funcionamento correto do motor de passo responsável pela dispensa da ração.

Procedimento: O motor foi programado para realizar rotações de 360° nos sentidos horário e anti-horário, com intervalos de 2 segundos entre as rotações.

Resultados: O motor de passo demonstrou operação suave e precisa em ambos os sentidos, confirmando sua aptidão para a funcionalidade de alimentação automática. A sequência de ativação das bobinas (in1, in2, in3, in4) foi validada.

4.2.2 **Teste do Módulo RTC DS3231**

Este teste visou validar a leitura e a precisão do relógio de tempo real (RTC).

Procedimento: O módulo RTC foi inicializado e a comunicação serial foi estabelecida para exibir a data e hora atuais. Foi incluída uma rotina para detectar a perda de energia do RTC e ajustar a hora manualmente (para 28/04/2025, 12:00:00) em caso de anomalia.

Resultados: O RTC operou conforme o esperado, mantendo a contagem do tempo de forma precisa. A funcionalidade de ajuste em caso de perda de energia foi confirmada, garantindo a continuidade das programações de alimentação e iluminação.

4.2.3 **Teste do Display LCD 16x2 com I2C**

O objetivo deste teste foi verificar a correta inicialização e exibição de mensagens no display LCD.

Procedimento: O display foi inicializado, o backlight ativado e mensagens de teste ("Teste do LCD" e "Linha inferior") foram escritas nas duas linhas do display.

Resultados: O display LCD funcionou perfeitamente, exibindo as mensagens de teste de forma clara. A comunicação via I2C foi estabelecida com sucesso, comprovando a integração do display ao sistema.

Este teste teve como objetivo verificar a correta inicialização e controle da fita de LED RGB, essencial para a funcionalidade de iluminação do aquário.

4.2.4 **Teste da Fita de LED WS2812B**

Este teste teve como objetivo verificar a correta inicialização e controle da fita de LED RGB, essencial para a funcionalidade de iluminação do aquário.

Procedimento: O sistema foi programado para inicializar a fita de LED. Foi enviada a cor CRGB::White (branco) para todos os LEDs, permanecendo acesa por 3 segundos. Em seguida, a cor foi alterada para CRGB::Red, permanecendo por 3 segundos. Finalmente, os LEDs foram desligados (FastLED.clear()), permanecendo por 3 segundos, e o ciclo se repetiu.

Resultados: A fita de LED WS2812B respondeu corretamente aos comandos, exibindo as cores programadas e sendo desligada de forma eficaz. Isso confirmou a correta comunicação com a biblioteca FastLED e a funcionalidade da iluminação.

4.2.5 **Teste da Interface I2C (Geral)**

Este teste visou confirmar a funcionalidade do barramento I2C, que é crucial para a comunicação entre o ESP32 e componentes como o display LCD e o módulo RTC.

Procedimento: O ESP32 foi programado para escanear o barramento I2C em busca de endereços de dispositivos. Qualquer endereço I2C encontrado foi impresso no monitor serial.

Resultados: O scanner I2C detectou com sucesso os endereços do display LCD (0x27) e do RTC (0x68 ou 0x57), confirmando a integridade e a funcionalidade do barramento I2C para a comunicação com múltiplos dispositivos.

4.2.6 **Teste da Página Web (Servidor Web AP)**

Este teste teve como objetivo verificar a capacidade do ESP32 de atuar como um Ponto de Acesso (AP) e hospedar uma página web funcional para configuração e monitoramento.

Procedimento: O ESP32 foi configurado para criar uma rede Wi-Fi (SSID: "AQUARIO", Senha: "aquario123"). Um dispositivo externo (smartphone ou computador) foi conectado a essa rede Wi-Fi. No navegador do dispositivo externo, foi acessado o endereço IP do ESP32 (impresso no monitor serial após a inicialização do AP). A interface web (paginaWeb()) foi carregada, e as opções de seleção de peixe e ambiente de iluminação foram testadas. O botão "Confirmar" foi acionado para verificar o envio das configurações para o ESP32.

Resultados: O ESP32 criou a rede Wi-Fi com sucesso e a página web foi acessível e renderizada corretamente no navegador. As seleções na página foram enviadas ao ESP32, que processou e atualizou as variáveis de configuração (peixeSelecionado e modoIluminacao), conforme verificado no monitor serial e pelas mudanças de comportamento do sistema (cor do LED, número de alimentações). Isso valida a funcionalidade da interface web como meio de interação remota.

4.2.7 **Teste do Sensor de Temperatura DS18B20**

Este teste visou validar a correta leitura de temperatura pelo sensor DS18B20.

Procedimento: O sensor DS18B20 foi conectado ao pino OneWire (GPIO 32) do ESP32. O sistema foi programado para solicitar leituras de temperatura em intervalos regulares. As leituras de temperatura em graus Celsius foram exibidas no

monitor serial e no display LCD. A reação do sensor a variações de temperatura (ex: contato com a mão, gelo) foi observada.

Resultados: O sensor DS18B20 forneceu leituras de temperatura precisas e consistentes, que foram corretamente exibidas no display LCD e no monitor serial. O sistema demonstrou a capacidade de detectar variações de temperatura, confirmando a funcionalidade do sensor e sua integração. Mensagens de erro para sensor desconectado também foram testadas e validadas.

4.3 TESTES DE FUNCIONALIDADE INTEGRADA

Após a validação dos componentes individuais, foram realizados testes para verificar o desempenho das funcionalidades principais em conjunto.

4.3.1 **Alimentador Automático de Peixes**

Procedimento: O sistema foi configurado com horários de alimentação (ex: 20:30 e 08:30) e monitorado para observar o acionamento do motor de passo nos horários programados. A interação do sistema com o RTC e a atualização do display LCD com os próximos horários foram verificadas.

Resultados: A alimentação ocorreu nos horários definidos, com o motor de passo dispensando a ração de forma consistente. O display atualizou corretamente os horários e o status da alimentação. A lógica de reset da alimentação à meia-noite também foi validada.

4.3.2 **Controle de Iluminação Automática**

Procedimento: Foram definidos horários para ligar (18:00) e desligar (06:00) a fita de LED RGB. O sistema foi observado para confirmar o acionamento da iluminação nos momentos programados.

Resultados: A iluminação foi ligada e desligada precisamente nos horários configurados, demonstrando o controle eficaz do sistema sobre o ciclo de luz do aquário.

4.3.3 **Medidor de Temperatura e Exibição**

Procedimento: O sensor de temperatura DS18B20 foi utilizado para coletar

leituras da água. Essas leituras foram exibidas no display LCD e monitoradas via serial.

Resultados: O sistema exibiu a temperatura da água em tempo real no display LCD e no monitor serial, com precisão adequada para o monitoramento do ambiente aquático. Falhas na conexão do sensor foram corretamente identificadas e sinalizadas.

4.4 CONSIDERAÇÕES DE DESEMPENHO

Durante os testes, o sistema demonstrou estabilidade e responsividade adequadas para um aquário doméstico. O consumo de energia do ESP32, conforme esperado, mostrou-se eficiente para aplicações contínuas. Não foram observados atrasos significativos na atualização de dados ou na execução dos comandos. Para atingir a performance desejada e otimizar os recursos do microcontrolador, diversas estratégias de otimização foram aplicadas ao longo do desenvolvimento do projeto. O Quadro 1 sumariza as principais áreas otimizadas e as abordagens implementadas para garantir a eficiência do sistema.

Figura 1 — Resumo das Áreas Otimizadas e Otimizações Aplicadas.

✅ **Resumo das Áreas Otimizadas**

Seção	Otimização Aplicada
Tempo de execução	`millis()` em vez de `delay()`
Comunicação	I2C e OneWire otimizam pinos e velocidade
Motor de passo	Controle com flags e desligamento após uso
Iluminação	LED só aceso quando necessário
Página Web	HTML direto e fetch rápido (sem redirecionamento)
Sensores	Leitura direta e mínima
Memória	Matrizes fixas e uso de `char[]` para menos RAM
Serial	Desativado para consumo quando não usado
Bibliotecas	Todas as bibliotecas usadas são leves e compatíveis

Fonte: Os autores (2025).

As otimizações implementadas, detalhadas no Quadro 1, foram cruciais para assegurar o funcionamento robusto e eficiente do sistema. A utilização de millis() em vez de delay(), por exemplo, permite que o microcontrolador execute outras tarefas enquanto aguarda, evitando bloqueios na execução do código. A comunicação otimizada via I2C e OneWire minimiza o uso de pinos e melhora a velocidade de troca de dados. No controle do motor de passo, a gestão por flags e o desligamento após o uso contribuem para a economia de energia. A iluminação, por sua vez, é ativada apenas quando necessário, reduzindo o consumo. A página web foi projetada com HTML direto e uso de fetch para respostas rápidas, eliminando redirecionamentos desnecessários. Sensores têm leitura direta e mínima para eficiência, enquanto o uso de matrizes fixas e char[] otimiza a memória RAM. Além disso, a comunicação serial é desativada quando não utilizada, e apenas bibliotecas leves e compatíveis foram selecionadas.

5 **CONCLUSÃO**

Este trabalho demonstrou com êxito a aplicação dos princípios de Sistemas Ciberfísicos (CPS) e da Internet das Coisas (IoT) no contexto da automação de aquários domésticos. Através do desenvolvimento do sistema "Aquário Automatizado", foi possível comprovar a viabilidade de um ambiente de cuidado aos animais aquáticos que transcende as limitações da intervenção manual. A integração harmoniosa entre hardware e software, aliada à capacidade de conectividade e controle remoto, resultou em um sistema robusto e eficiente, capaz de otimizar significativamente a rotina de manutenção do aquário.

Os objetivos propostos foram integralmente alcançados. O sistema demonstrou a capacidade de controlar automaticamente a alimentação dos peixes, monitorar e exibir a temperatura da água em tempo real, gerenciar a iluminação de forma programada e oferecer uma interface remota para interação do usuário. A escolha do microcontrolador ESP32 revelou-se adequada, proporcionando a conectividade Wi-Fi necessária e um desempenho satisfatório para as funcionalidades implementadas. A precisão do módulo RTC DS3231 e a eficiência da comunicação I2C foram elementos cruciais para a estabilidade e o design compacto do projeto.

A validação dos módulos por meio de testes isolados e a integração do sistema confirmaram a funcionalidade e a confiabilidade das soluções propostas, consolidando o potencial da automação para proporcionar maior conforto e segurança aos animais e praticidade aos seus cuidadores. A organização da equipe e a utilização de ferramentas de versionamento, como o GitHub, foram fundamentais para a colaboração e o gerenciamento eficaz do desenvolvimento.

Em termos de perspectivas futuras, o projeto possui um vasto campo para expansão. A incorporação de sensores adicionais, como medidores de pH ou de nível de água, poderia enriquecer ainda mais o monitoramento ambiental do aquário. Adicionalmente, a implementação de uma conexão com serviços de nuvem permitiria a coleta e análise de dados históricos, aprimorando a inteligência do sistema e possibilitando recursos como notificações proativas ou ajustes adaptativos baseados em padrões de comportamento dos peixes ou condições ambientais. Tais aprimoramentos solidificariam o "Aquário Automatizado" como uma solução ainda mais completa e adaptável às necessidades dos usuários e do ecossistema aquático.

REFERÊNCIAS

ADAFRUIT. RTClib: Real Time Clock library. Adafruit. Disponível em: https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/overview. Acesso em: 14 jun. 2025.

ARDUINO: ESP32 with Arduino IDE - Getting Started. Random Nerd Tutorials. Disponível em: https://randomnerdtutorials.com/getting-started-with-esp32/. Acesso em: 14 jun. 2025.

FRANCISCO THENÓRIO, Iberê. **Alimentador automático de peixes**. CANAL MANUAL DO MUNDO. Disponível em: https://www.youtube.com/watch?v=qaEer1_0UBc&t=1s. Acesso em: 15 mai. 2025.

MILES, BURTON. **Arduino-Temperature-Control-Library**. Github. Disponível em: https://github.com/milesburton/Arduino-Temperature-Control-Library. Acesso em: 24 mai. 2025.

GLOSSÁRIO

Expressão	Descrição
1-Wire	Protocolo de comunicação serial que permite a comunicação bidirecional de dados em um único fio, além do terra. É comumente usado para sensores, como o DS18B20.
Arduino Uno	Plataforma de prototipagem eletrônica de código aberto, baseada em hardware e software fáceis de usar. Mencionada no projeto como uma alternativa ao ESP32, sendo o ESP32 superior para aplicações conectadas.
Atuadores	Componentes que convertem um sinal elétrico em movimento ou outra ação física. No projeto, incluem o Motor de Passo e a fita de LED RGB.
Driver ULN2003	Circuito integrado amplificador de corrente que é usado para controlar motores de passo, como o 28BYJ-48.
DS18B20	Sensor de temperatura digital de alta precisão que opera com o protocolo 1-Wire. Utilizado no projeto para monitorar a temperatura da água do aquário.
ESP32	Microcontrolador com conectividade Wi-Fi e Bluetooth integradas, conhecido por seu baixo consumo de energia e ampla compatibilidade com sensores e atuadores. É o cérebro do sistema automatizado do aquário.
GPIO	Abreviação de General-Purpose Input/Output (Entrada/Saída de Propósito Geral). Pinos digitais em um microcontrolador que podem ser configurados para funcionar como entrada/saída.
Hardware	Refere-se aos componentes físicos do sistema, como o microcontrolador, sensores, atuadores e display.
HTTP/HTTPS	Protocolos de comunicação usados para a transferência de dados pela internet. O HTTPS é a versão segura do HTTP. Utilizados para o envio de dados em tempo real e controle remoto.
I2C	Protocolo de comunicação serial bidirecional que permite a comunicação entre múltiplos dispositivos com apenas dois fios (SDA e SCL). Adotado no projeto por sua eficiência e economia de pinos no microcontrolador.

IoT	Conceito que descreve a rede de objetos físicos que incorporam sensores, software e outras tecnologias para se conectar e trocar dados com outros dispositivos e sistemas pela internet[cite: 14]. O projeto do aquário automatizado se enquadra nos princípios da IoT
LCD 16x2	Tipo de display de cristal líquido que pode exibir 16 caracteres por 2 linhas. Usado para exibir informações importantes do aquário, como horário e temperatura.
28BYJ-48	Tipo de motor elétrico que divide uma rotação completa em um número de passos discretos, permitindo um controle preciso da posição. Usado para a alimentação automática dos peixes.
RTC DS3231	Módulo de relógio de tempo real altamente preciso e estável termicamente, que mantém a contagem do tempo (data e hora) mesmo quando o microcontrolador está desligado. Escolhido no projeto por sua precisão.
Sensores	Dispositivos que detectam e respondem a algum tipo de entrada do ambiente físico, como temperatura ou luz. No projeto, inclui DS18B20.
CPS	Sistemas Ciberfísicos que integram capacidades computacionais e de comunicação com processos físicos. Eles permitem a interação em tempo real entre o mundo físico e o cibernético. O aquário automatizado é um exemplo de CPS.
Software	Refere-se aos programas e códigos que controlam o hardware do sistema. No projeto, inclui o código programado no ESP32.
Wi-Fi	Tecnologia de rede sem fio que permite que dispositivos se conectem à internet ou a outras redes sem a necessidade de cabos. O ESP32 possui conectividade Wi-Fi integrada.

APÊNDICE A — CÓDIGO-FONTE COMPLETO DO AQUÁRIO

```
// ============================
// PROJETO: Aquário Inteligente com ESP32
// Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
// ============================

// Bibliotecas necessárias para cada componente
#include <WiFi.h> // Comunicação Wi-Fi
#include <WebServer.h> // Criação de servidor web
#include <Wire.h> // Comunicação I2C
#include <RTClib.h> // Relógio de tempo real (RTC)
#include <LiquidCrystal_I2C.h> // Display LCD via I2C
#include <OneWire.h> // Comunicação OneWire
#include <DallasTemperature.h> // Sensor de temperatura DS18B20
#include <FastLED.h> // Controle da fita de LED WS2812B
#include <Stepper.h> // Controle do motor de passo

// ============================
// CONFIGURAÇÃO DE REDE Wi-Fi (modo AP)
// ============================
const char* ssid = "AQUARIO";
const char* senha = "aquario123";
WebServer server(80); // Cria um servidor na porta 80

// ============================
// INSTÂNCIAS DOS COMPONENTES
// ============================
LiquidCrystal_I2C lcd(0x27, 16, 2); // Endereço I2C 0x27, display 16x2
RTC_DS3231 rtc; // Relógio de tempo real DS3231

// Motor de passo 28BYJ-48 (sequência de pinos)
#define in1 25
#define in2 26
#define in3 27
#define in4 14
Stepper motor(2048, in1, in3, in2, in4); // 2048 passos por volta
```

```
// Sensor de temperatura DS18B20
#define ONE_WIRE_BUS 32
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

// Fita de LED WS2812B
#define DATA_PIN 5
#define NUM_LEDS 30
CRGB leds[NUM_LEDS];

// ============================
// VARIÁVEIS GLOBAIS
// ============================
String peixeSelecionado = "betta";
float temperaturaAtual = 0.0;
bool motorAtivadoHoje = false;
String modoIluminacao = "com_luz";

int alimentacoesDia = 2;
int horarios[3][2] = {{8, 0}, {20, 0}, {23, 0}}; // Horários de alimentação

unsigned long tempoAnterior = 0;
int telaAtual = 0; // Tela atual exibida no LCD

// ============================
// CONFIGURAÇÃO DO PEIXE ESCOLHIDO
// ============================
void configurarPeixe(String peixe) {
  peixeSelecionado = peixe;

  // Configura cor da fita de LED e número de alimentações
  if (peixe == "betta") {
    fill_solid(leds, NUM_LEDS, CRGB::LightBlue);
    alimentacoesDia = 2;
  } else if (peixe == "neon") {
    fill_solid(leds, NUM_LEDS, CRGB::Green);
    alimentacoesDia = 2;
  } else if (peixe == "oscar") {
```

```
      fill_solid(leds, NUM_LEDS, CRGB::Orange);
      alimentacoesDia = 3;
    }
    FastLED.show(); // Atualiza a cor dos LEDs
  }

  // ============================
  // VERIFICA SE A TEMPERATURA ESTÁ IDEAL
  // ============================
  bool temperaturaIdeal() {
      if (peixeSelecionado == "betta") return temperaturaAtual >= 26 &&
temperaturaAtual <= 28;
      if (peixeSelecionado == "neon") return temperaturaAtual >= 22 &&
temperaturaAtual <= 26;
      if (peixeSelecionado == "oscar") return temperaturaAtual >= 25 &&
temperaturaAtual <= 27;
    return true;
  }

  // ============================
  // RETORNA STATUS DA TEMPERATURA
  // ============================
  String statusTemperatura() {
    if (peixeSelecionado == "betta") {
      if (temperaturaAtual < 26) return "Baixa";
      if (temperaturaAtual > 28) return "Alta";
    } else if (peixeSelecionado == "neon") {
      if (temperaturaAtual < 22) return "Baixa";
      if (temperaturaAtual > 26) return "Alta";
    } else if (peixeSelecionado == "oscar") {
      if (temperaturaAtual < 25) return "Baixa";
      if (temperaturaAtual > 27) return "Alta";
    }
    return "OK";
  }

  // ============================
  // CALCULA PRÓXIMA REFEIÇÃO COM BASE NA HORA ATUAL
```

```
// ============================
String proximaRefeicao(DateTime agora) {
  for (int i = 0; i < alimentacoesDia; i++) {
        if (agora.hour() < horarios[i][0] || (agora.hour() == horarios[i][0] &&
agora.minute() < horarios[i][1])) {
      char buffer[6];
      sprintf(buffer, "%02d:%02d", horarios[i][0], horarios[i][1]);
      return String(buffer);
    }
  }
  // Se já passou todas, retorna o primeiro horário do próximo dia
  char buffer[6];
  sprintf(buffer, "%02d:%02d", horarios[0][0], horarios[0][1]);
  return String(buffer);
}

// ============================
// ATUALIZA O DISPLAY LCD DE ACORDO COM A TELA
// ============================
void atualizarLCD(DateTime agora) {
  char linha1[17], linha2[17];
  lcd.clear();

  if (telaAtual == 0) {
    sprintf(linha1, "Hora: %02d:%02d", agora.hour(), agora.minute());
    sprintf(linha2, "Prox: %s", proximaRefeicao(agora).c_str());
  } else if (telaAtual == 1) {
    sprintf(linha1, "Temp: %.1fC", temperaturaAtual);
    sprintf(linha2, "Status: %s", statusTemperatura().c_str());
  } else {
    sprintf(linha1, "Peixe: %s", peixeSelecionado.c_str());
    if (peixeSelecionado == "betta") sprintf(linha2, "Luz: Azul Claro");
    if (peixeSelecionado == "neon") sprintf(linha2, "Luz: Verde");
    if (peixeSelecionado == "oscar") sprintf(linha2, "Luz: Laranja");
  }

  lcd.setCursor(0, 0); lcd.print(linha1);
  lcd.setCursor(0, 1); lcd.print(linha2);
```

```
}

// ============================
// CONSTRÓI A PÁGINA WEB HTML
// ============================
void paginaWeb() {
  String html = R"rawliteral(
      <!DOCTYPE html><html><head><meta charset='utf-8'><title>Aquário Inteligente</title><style>
       body {font-family: Arial; background: #eef; max-width: 700px; margin: auto; padding: 20px;}
      h2 {color: #003366;}
       .card {background: white; padding: 10px; border-radius: 10px; margin-top: 10px; box-shadow: 0 0 6px #999;}
      </style></head><body>
      <h2>Preferências de cada peixe</h2>
        <div class='card'> **Betta**<br>Luz: Azul Claro<br>Temperatura: 26–28°C<br>Alimentação: 2x/dia<br>*Betta prefere luz suave, simulando águas calmas e tropicais.*</div>
        <div class='card'> **Neon Tetra**<br>Luz: Verde<br>Temperatura: 22–26°C<br>Alimentação: 2x/dia<br>*A luz verde simula vegetação densa, ambiente natural para Neon.*</div>
        <div class='card'> **Oscar**<br>Luz: Laranja<br>Temperatura: 25–27°C<br>Alimentação: 3x/dia<br>*Oscar vive bem com luz quente, típica de águas barrentas.*</div>
      <hr>
      <h2>Configurações</h2>
      <label>Peixe selecionado:</label><br>
      <select id='peixe'>
       <option value='betta'>Betta</option>
       <option value='neon'>Neon Tetra</option>
       <option value='oscar'>Oscar</option>
      </select><br><br>
      <label>Ambiente do aquário:</label><br>
      <select id='ambiente'>
       <option value='com_luz'>Recebe luz natural (ex: janela)</option>
       <option value='sem_luz'>Ambiente fechado (ex: quarto)</option>
      </select><br><br>
      <button onclick='enviar()'>Confirmar</button>
```

```
            <p id='resposta'></p>
            <script>
              function enviar() {
                let peixe = document.getElementById('peixe').value;
                let ambiente = document.getElementById('ambiente').value;
                fetch('/config?peixe=' + peixe + '&ambiente=' + ambiente)
                                                    .then(r => r.text()).then(d => 
document.getElementById('resposta').innerText = d);
              }
            </script>
          </body></html>)rawliteral";

          server.send(200, "text/html", html);
        }

        // ============================
        // SETUP: INICIALIZAÇÃO DOS COMPONENTES
        // ============================
        void setup() {
          Serial.begin(115200);

          // Cria rede Wi-Fi no modo ponto de acesso (AP)
          WiFi.softAP(ssid, senha);
          Serial.println("Rede criada!");
          Serial.println(WiFi.softAPIP());

          // Roteamento das páginas do servidor
          server.on("/", paginaWeb);
          server.on("/config", HTTP_GET, []() {
            if (server.hasArg("peixe")) configurarPeixe(server.arg("peixe"));
            if (server.hasArg("ambiente")) modoIluminacao = server.arg("ambiente");
                server.send(200, "text/plain", "Configurações atualizadas!\nPeixe: " + 
peixeSelecionado + "\nAmbiente: " + modoIluminacao);
          });
          server.begin();

          // Inicialização dos módulos
          Wire.begin(21, 22); // I2C nos pinos padrão do ESP32
          rtc.begin();
```

```
  lcd.init(); lcd.backlight();
  sensors.begin();
  motor.setSpeed(10);
  FastLED.addLeds<WS2812B, DATA_PIN, GRB>(leds, NUM_LEDS);

  configurarPeixe(peixeSelecionado);
 }

 // ============================
 // LOOP PRINCIPAL
 // ============================
 void loop() {
  server.handleClient(); // Lida com requisições da página
  DateTime agora = rtc.now(); // Obtém hora atual do RTC
  sensors.requestTemperatures(); // Solicita leitura de temperatura
  temperaturaAtual = sensors.getTempCByIndex(0); // Lê temperatura

  // Alterna telas no LCD a cada 5 segundos
  if (millis() - tempoAnterior > 5000) {
   telaAtual = (telaAtual + 1) % 3;
   tempoAnterior = millis();
   atualizarLCD(agora);
  }

  // Verifica se é hora de alimentar o peixe
  for (int i = 0; i < alimentacoesDia; i++) {
      if (agora.hour() == horarios[i][0] && agora.minute() == horarios[i][1] &&
agora.second() == 0 && !motorAtivadoHoje) {
     motor.step(2048 * 0.25); // Gira 1/4 de volta
     digitalWrite(in1, LOW); digitalWrite(in2, LOW);
     digitalWrite(in3, LOW); digitalWrite(in4, LOW);
     motorAtivadoHoje = true;
   }
  }

  // Reset diário da flag do motor
  if (agora.hour() == 0 && agora.minute() == 0 && agora.second() == 0) {
   motorAtivadoHoje = false;
  }
```

```
  // Controle de iluminação conforme horário e ambiente
  if (modoIluminacao == "com_luz") {
    if (agora.hour() >= 8 && agora.hour() < 20) FastLED.show();
    else FastLED.clear();
  } else {
    FastLED.show();
  }

  delay(500); // Pequeno delay para evitar uso excessivo da CPU
}
```

APÊNDICE B — TODOS OS CÓDIGOS DOS TESTES ISOLADOS

```
/*
 TESTE DO MOTOR DE PASSO 28BYJ-48 COM ESP32
Grupo: Matheus Muller, Germano Lagana, Vinicius Padilha
Objetivo: Verificar o funcionamento do motor de passo utilizando as portas digitais do ESP32.
  OBS: Certifique-se de que os fios do driver ULN2003 estão corretamente conectados às portas digitais indicadas.
*/

#include <Stepper.h>

// Define o número de passos por volta do motor (para o 28BYJ-48 com redutor: 2048 passos)
#define PASSOS_POR_REVOLUCAO 2048

// Define os pinos conectados ao driver do motor (ULN2003)
#define IN1 25
#define IN2 26
#define IN3 27
#define IN4 14

// Cria o objeto do motor de passo com os pinos definidos
Stepper motor(PASSOS_POR_REVOLUCAO, IN1, IN3, IN2, IN4);

void setup() {
  // Inicia a comunicação serial para feedback no monitor
  Serial.begin(115200);
  Serial.println("Iniciando teste do motor de passo...");

  // Define a velocidade do motor (em rotações por minuto)
  motor.setSpeed(10);
}

void loop() {
  Serial.println("Girando no sentido horário 1/2 volta...");
    motor.step(PASSOS_POR_REVOLUCAO / 2); // Gira 1024 passos (meia volta)
```

```
  delay(2000);

  Serial.println("Girando no sentido anti-horário 1/2 volta...");
   motor.step(-PASSOS_POR_REVOLUCAO / 2); // Gira -1024 passos (meia volta)
  delay(2000);
}

/*
TESTE – MÓDULO RTC DS3231
Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
Este teste inicializa o módulo de relógio em tempo real (RTC) DS3231,
sincroniza o horário (se necessário) e imprime continuamente a data e a
hora atual no monitor serial.
*/

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

RTC_DS3231 rtc;

void setup() {
  Serial.begin(115200);
  delay(1000);

  // Inicia comunicação I2C nos pinos padrão do ESP32
  Wire.begin(21, 22);

  // Inicia o RTC
  if (!rtc.begin()) {
    Serial.println("✖ ERRO: RTC não encontrado!");
    while (1);
  }

  // Caso a hora ainda não tenha sido configurada:
  // rtc.adjust(DateTime(F(__DATE__), F(__TIME__))); // Configura com a hora do computador

  Serial.println("✓ RTC iniciado com sucesso!");
```

```
}

void loop() {
  DateTime agora = rtc.now();

  Serial.print("Data: ");
  Serial.print(agora.day());
  Serial.print("/");
  Serial.print(agora.month());
  Serial.print("/");
  Serial.print(agora.year());

  Serial.print(" | Hora: ");
  Serial.print(agora.hour());
  Serial.print(":");
  Serial.print(agora.minute());
  Serial.print(":");
  Serial.println(agora.second());

  delay(1000);
}

/*
TESTE – DISPLAY LCD 16x2 I2C
Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
Este teste inicializa o display LCD 16x2 com interface I2C e exibe duaslinhas
de texto fixo. Deve-se ver a mensagem "Aquário Inteligente" na primeiralinha
E "Display Funcionando" na segunda.
*/

#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// ======= CONFIGURAÇÃO DO DISPLAY =======
LiquidCrystal_I2C lcd(0x27, 16, 2); // Endereço I2C pode ser 0x27 ou 0x3F.
Verifique se necessário!

void setup() {
```

```
  // Inicializa a comunicação I2C com ESP32 nos pinos padrão (21=SDA, 22=SCL)
  Wire.begin(21, 22);

  // Inicia o display e acende a luz de fundo
  lcd.init();
  lcd.backlight();

  // Limpa o display
  lcd.clear();

  // Define o que será exibido
  lcd.setCursor(0, 0); // Coluna 0, linha 0
  lcd.print("Aquario Inteligente");

  lcd.setCursor(0, 1); // Coluna 0, linha 1
  lcd.print("Display funcionando");

  // Mensagem no serial para confirmação
  Serial.begin(115200);
  Serial.println("Display LCD iniciado com sucesso!");
}

void loop() {
  // Nada a repetir neste teste
}

/*
TESTE DA FITA DE LED WS2812B COM ESP32
Grupo: Matheus Muller, Germano Lagana, Vinicius Padilha
Objetivo: Verificar o funcionamento da fita de LED (WS2812B) com controle de cor e brilho via ESP32.
OBS: Certifique-se de que:
- O pino de dados (Data IN) da fita está conectado ao pino 5 do ESP32;
- A alimentação está adequada (idealmente com fonte externa 5V se tiver muitos LEDs);
- GND do ESP32 e da fonte externa estão interligados.
*/
```

```
#include <FastLED.h>

// Define o pino de dados e o número de LEDs
#define DATA_PIN 5
#define NUM_LEDS 30

// Cria o array que representa os LEDs
CRGB leds[NUM_LEDS];

void setup() {
  // Inicializa a comunicação serial
  Serial.begin(115200);
  Serial.println("Iniciando teste da fita de LED...");

  // Inicializa os LEDs (modelo WS2812B com ordem de cores GRB)
  FastLED.addLeds<WS2812B, DATA_PIN, GRB>(leds, NUM_LEDS);
  FastLED.setBrightness(100); // Define brilho (0 a 255)
}

void loop() {
  Serial.println("Cor: Vermelho");
  fill_solid(leds, NUM_LEDS, CRGB::Red);
  FastLED.show();
  delay(1000);

  Serial.println("Cor: Verde");
  fill_solid(leds, NUM_LEDS, CRGB::Green);
  FastLED.show();
  delay(1000);

  Serial.println("Cor: Azul");
  fill_solid(leds, NUM_LEDS, CRGB::Blue);
  FastLED.show();
  delay(1000);

  Serial.println("Desligando LEDs...");
  fill_solid(leds, NUM_LEDS, CRGB::Black);
```

```
  FastLED.show();
  delay(1000);
}

/*
Teste da Interface I2C – Scanner de dispositivos
Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
Objetivo: Verificar se o barramento I2C está funcionando corretamente e
quais dispositivos estão conectados.
*/

#include <Wire.h>

void setup() {
  Serial.begin(115200); // Inicia comunicação serial
  Wire.begin(21, 22); // Pinos padrão do ESP32: SDA = 21, SCL = 22
  delay(1000);
  Serial.println("    Iniciando scanner I2C...");
}

void loop() {
  byte count = 0;

  Serial.println("    Varredura dos dispositivos I2C...");
  for (byte address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if (Wire.endTransmission() == 0) {
      Serial.print("✅ Dispositivo encontrado no endereço 0x");
      if (address < 16) Serial.print("0");
      Serial.print(address, HEX);
      Serial.println();
      count++;
      delay(10);
    }
  }

  if (count == 0) {
    Serial.println("⚠ Nenhum dispositivo I2C encontrado.");
```

```
  } else {
    Serial.print("    Total de dispositivos encontrados: ");
    Serial.println(count);
  }

  Serial.println("    Aguardando 5 segundos para nova varredura...\n");
  delay(5000); // Aguarda antes de repetir
}

/*
  TESTE – PÁGINA WEB COM INFORMAÇÕES DOS PEIXES
  Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
  Este teste cria uma rede Wi-Fi local chamada "AQUARIO_TESTE".
  Ao acessar pelo navegador, o usuário verá as preferências de cada peixe:
  - Luz ideal
  - Faixa de temperatura
  - Frequência de alimentação
*/

#include <WiFi.h>
#include <WebServer.h>

// ======= CONFIGURAÇÃO DO Wi-Fi =======
const char* ssid = "AQUARIO_TESTE"; // Nome da rede
const char* senha = "aquario123"; // Senha da rede

WebServer server(80); // Servidor HTTP na porta 80

// ======= HTML da Página com as Informações dos Peixes =======
void paginaWeb() {
  String html = R"rawliteral(
        <!DOCTYPE  html><html><head><meta  charset='utf-8'><title>Aquário
Inteligente</title><style>
         body {font-family: Arial; background: #eef; max-width: 700px; margin: auto;
padding: 20px;}
        h2 {color: #003366;}
         .card {background: white; padding: 10px; border-radius: 10px; margin-top:
10px; box-shadow: 0 0 6px #999;}
```

```
    </style></head><body>
    <h2>Preferências de cada peixe</h2>
      <div class='card'> <b>Betta</b><br>Luz: Azul Claro<br>Temperatura: 26–28°C<br>Alimentação: 2x/dia<br><i>Betta prefere luz suave, simulando águas calmas e tropicais.</i></div>
      <div class='card'> <b>Neon Tetra</b><br>Luz: Verde<br>Temperatura: 22–26°C<br>Alimentação: 2x/dia<br><i>A luz verde simula vegetação densa, ambiente natural para Neon.</i></div>
      <div class='card'> <b>Oscar</b><br>Luz: Laranja<br>Temperatura: 25–27°C<br>Alimentação: 3x/dia<br><i>Oscar vive bem com luz quente, típica de águas barrentas.</i></div>
    </body></html>)rawliteral";

  server.send(200, "text/html", html); // Envia a página ao navegador
}

void setup() {
  Serial.begin(115200);
  WiFi.softAP(ssid, senha); // Cria rede Wi-Fi
  Serial.println("Rede criada!");
  Serial.print("IP: ");
  Serial.println(WiFi.softAPIP()); // Mostra IP da rede

  server.on("/", paginaWeb); // Define a página principal
  server.begin(); // Inicia o servidor
}

void loop() {
  server.handleClient(); // Atende as requisições HTTP
}

/*
TESTE – SENSOR DE TEMPERATURA DS18B20
Integrantes: Matheus Muller, Germano Lagana e Vinicius Padilha
Este teste lê a temperatura da água do aquário utilizando o sensor
DS18B20.
A leitura é feita via protocolo OneWire e exibida no Monitor Serial.
*/
```

```
#include <OneWire.h>
#include <DallasTemperature.h>

// ======= CONFIGURAÇÃO DO PINO DO SENSOR =======
#define ONE_WIRE_BUS 32 // Pino digital onde o sensor está conectado (ajuste se necessário)

OneWire oneWire(ONE_WIRE_BUS); // Inicializa comunicação OneWire
DallasTemperature sensors(&oneWire); // Usa biblioteca Dallas para ler o DS18B20

void setup() {
  Serial.begin(115200); // Inicia comunicação serial
  sensors.begin(); // Inicia o sensor
  Serial.println("Teste do Sensor de Temperatura iniciado!");
}

void loop() {
  sensors.requestTemperatures(); // Solicita leitura ao sensor
   float temperatura = sensors.getTempCByIndex(0); // Lê a temperatura em Celsius

  // Verifica se a leitura é válida
  if (temperatura == DEVICE_DISCONNECTED_C) {
    Serial.println("Erro: Sensor desconectado ou leitura inválida!");
  } else {
    Serial.print("Temperatura atual: ");
    Serial.print(temperatura);
    Serial.println(" °C");
  }

  delay(2000); // Aguarda 2 segundos antes da próxima leitura
}
```