UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
ESCOLA AGRÍCOLA DE JUNDIAÍ
UNIDADE ACADÊMICA ESPECIALIZADA EM CIÊNCIAS AGRÁRIAS
CURSO TÉCNICO INTEGRADO EM INFORMÁTICA

Kedra pet: sistema de dispenser de ração controlado por plataforma web

Juan Gabriel Gomes

Macaíba-RN
DEZEMBRO/2024

Juan Gabriel Gomes

Kedra pet: sistema de dispenser de ração controlado por plataforma web

Trabalho de conclusão de curso apresentado ao curso Técnico em Informática da Escola Agrícola de Jundiaí da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de Técnico em Informática.

Tipo do Trabalho: Monografia

Orientador
Prof. Dr. Josenalde Barbosa de Oliveira

Escola Agrícola de Jundiaí - EAJ
Universidade Federal do Rio Grande do Norte - UFRN

Macaíba-RN
DEZEMBRO/2024

Trabalho de conclusão de curso sob o título *Kedra pet: sistema de dispenser de ração controlado por plataforma web* apresentado por Juan Gabriel Gomes e aceita pelo curso Técnico em Informática da Escola Agrícola de Jundiaí da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Prof. Dr. Josenalde Barbosa de Oliveira

Presidente

EAJ – Escola Agrícola de Jundiaí

UFRN – Universidade Federal do Rio Grande do Norte

Prof. Dr. Leonardo Rodrigues de Lima Teixeira

EAJ – Escola Agrícola de Jundiaí

UFRN – Universidade Federal do Rio Grande do Norte

Tec. Adm. Esp. Luiz Fernando Ferreira da Silva

EAJ – Escola Agrícola de Jundiaí

UFRN – Universidade Federal do Rio Grande do Norte

Macaíba-RN, 11 de dezembro de 2024.

Aos meus queridos avós, Orlando e Marisa, meus maiores apoiadores.

Agradecimentos

Primeiramente, gostaria de agradecer a Deus, meu maior apoiador e capacitador durante toda essa caminhada acadêmica e pessoal, jornada que proporcionou experiências imensuráveis para mim. Agradeço também a minha família, que estiveram comigo durante todo o processo, mesmo que distantes fisicamente, apoiando e incentivando-me a todo instante. Por fim, demonstro aqui minha gratidão aos meus caros amigos, que me acompanharam por quase três anos e contribuíram para a realização deste trabalho e conclusão deste curso técnico, e também aos professores que cederam do seu tempo e conhecimento para minha formação.

Qualquer tecnologia suficientemente avançada é indistinta de magia.

Arthur C. Clarke

Kedra pet: sistema de dispenser de ração controlado por plataforma web

Autor: Juan Gabriel Gomes
Orientador: Prof. Dr. Josenalde Barbosa de Oliveira

RESUMO

Este trabalho tem como principal objetivo o desenvolvimento de um sistema de dispenser de ração controlado por plataforma web, denominado Kedra pet. Esse mecanismo pode liberar o conteúdo armazenado nos horários e quantidades programadas pelo usuário. Será abordada a investigação realizada para a definição dos meios utilizados, testes e a construção do circuito eletrônico, bem como as documentações e estruturação do site, e comunicação entre a plataforma e o circuito. Por fim, será exposto o resultado final dos trabalhos e demonstração do funcionamento ideal da solução.

Palavras-chave: sistemas embarcados, desenvolvimento web, eletrônica, robótica.

Kedra pet: feed dispenser system controlled by web platform

Author: Juan Gabriel Gomes
Advisor: Prof. Dr. Josenalde Barbosa de Oliveira

ABSTRACT

The main objective of this work is to develop a feed dispenser system controlled by a web application, called Kedra pet. This mechanism can release the stored content at the times and in the quantities programmed by the user. It will be addressed the research carried out to define the means used, tests and the construction of the electronic circuit, as well as the documentation and structuring of the website, and communication between the platform and the circuit. Finally, it will be exposed the final result of the work and demonstrate the ideal functioning of the solution.

Keywords: embedded systems, web development, electronics, robotics.

Lista de figuras

Lista de abreviaturas e siglas

PIB - Produto interno bruto, p.13

HTML - *HyperText Markup Language*, p. 14

CSS - *Cascading Style Sheets*, p. 15

CRUD - *Create* (criar)*, Read* (ler)*, Update* (atualizar) *e Delete* (apagar), p. 36.

Sumário

1 Introdução

A tecnologia tem feito avanços cada vez mais significativos nas últimas décadas, em ritmo extremamente acelerado. Como exemplo, pode-se citar a Internet, criada em 1969 e popularizada na década de 90, que hoje contribui ativamente para o fenômeno da globalização e se faz presente na vida de quase 70% da população brasileira (OYADOMARI). Esses avanços auxiliam a sociedade moderna de inúmeras formas, principalmente promovendo mais comodidade e facilidade em pequenas tarefas diárias e atingindo diversas áreas, como por exemplo o mercado pet.

Atualmente, existem soluções que se propõem a facilitar alguns dos desafios enfrentados por tutores de pets, entre eles a alimentação disfuncional de seus animais domésticos. Essa pode ser causada por diversos fatores, entre eles: a falta de tempo hábil para depositar a ração diversas vezes no dia, e a irregularidade nas quantidades e horários necessários.

Pensando nisso, o propósito deste trabalho é desenvolver um sistema de dispenser de ração para pets, denominado Kedra pet. Essa solução será controlada por uma plataforma web, onde o usuário poderá cadastrar as alimentações diárias de seu animal inserindo os horários e as quantidades de cada uma delas. Baseado nessas informações, o dispenser abrirá no tempo indicado e irá despejar o conteúdo até atingir a quantidade definida.

1.1 Organização do trabalho

Inicialmente, será demonstrada a investigação e definição do problema, expondo as motivações que levaram a produção deste trabalho e um quadro geral do público que utiliza soluções semelhantes no dia a dia.

Em seguida, o texto abordará o detalhamento dos componentes escolhidos para o projeto, plataforma de testes, linguagens e paradigma de programação, entre outros. Partindo para o desenvolvimento do sistema, serão abordados os testes e a programação do projeto. Entre eles tem-se, simulações digitais utilizando

plataformas como Wowki[1], para teste do microcontrolador ESP-32 com o servomotor para simular o movimento do abrir/fechar do dispenser. Também as experimentações para o desenvolvimento do site com comunicação via websocket com o ESP32. Logo após, o trabalho se dedicará ao processo de criação do sistema, com descrição das etapas e principais desafios enfrentados na prática, e detalhamento de cada parte do projeto: back-end, front-end e programação do ESP-32.

Por fim, concluirá mostrando os principais resultados do Trabalho de Conclusão de Curso. O sistema eletrônico digital, responsável pelo funcionamento do dispenser, e a plataforma web responsável pelo controle do dispenser, o banco de dados utilizado, o servidor, bem como as documentações requeridas para o projeto do software, e os testes com a solução.

[1] https://wokwi.com/projects/new/esp32

2 Metodologia

2.1 Investigação e definição do problema

Segundo a ABINPET, a Associação Brasileira da Indústria de Produtos para Animais de Estimação, estima-se que existam um total de 167,6 milhões de pets no Brasil. Desses, 67,8 milhões são cães e 33,6 milhões são gatos. Tendo isso em mente, o segmento pet tem tudo para estar sempre em funcionamento e evolução, visto que possui um público fiel em constante crescimento. Esse mercado de produtos já representa 0,36% do PIB brasileiro (MEDEIROS, 2024). Baseando-se nesses dados, pode-se deduzir o significativo avanço da indústria, que passou a desenvolver soluções que facilitem as pequenas e rotineiras tarefas dos tutores.

Entre essas atividades cotidianas, destaca-se como uma das essenciais a alimentação dos pets, ou seja, depositar o alimento (geralmente ração) em horários específicos e quantidades adequadas para cada animal. Então surgem os alimentadores inteligentes, que se propõem a resolver uma dor de boa parte do público geral. Essas soluções visam sanar a dificuldade de estar presente em todos os momentos ideais para alimentar seus companheiros pet, que muitas vezes ficam sozinhos a maior parte do tempo enquanto seus tutores enfrentam longas jornadas de trabalho e uma dinâmica rotina urbanizada.

Em linhas gerais, esses produtos oferecem primordialmente a possibilidade de programar os horários e quantidades de cada alimentação diária dos animais. Os aparelhos promovem a segurança necessária aos tutores que seus companheiros de quatro patas estarão bem nutridos, independente de sua ausência durante períodos de curta ou média duração, dependendo da solução. Geralmente, são controlados por aplicativos móveis, pelos quais você configura e monitora as atividades realizadas pelo sistema físico do dispenser.

A seguir, nas figuras 1 e 2, é possível visualizar alguns exemplos de alimentadores inteligentes ou automáticos, já disponíveis para compra em uma variedade de lojas online e presenciais.

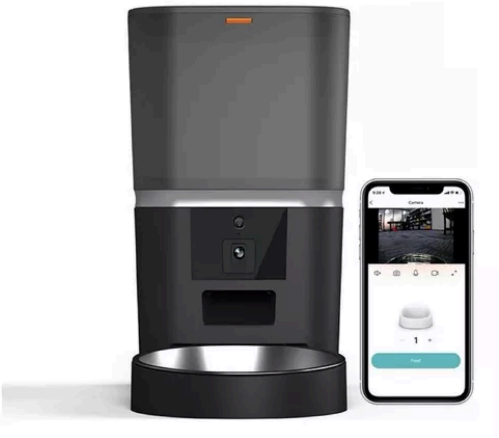


Figura 1 - Alimentador inteligente pet Tuya

Figura 2 - Alimentador automático pet Dogis

2.2 Proposta do projeto

Conforme o que foi apresentado anteriormente, a proposta deste Trabalho de Conclusão de Curso é desenvolver um sistema simplificado baseado nas soluções já comercializadas, utilizando os conhecimentos adquiridos durante o curso técnico de Informática. As áreas da robótica, eletrônica, programação orientada a objetos e

desenvolvimento web se farão essenciais nesse projeto.

A solução é composta por:

- plataforma web criada com *HyperText Markup Language* (HTML) e o *Cascading Style Sheets* (CSS).
- Banco de dados em nuvem controlado pelo servidor, na plataforma Firebase.
- Servidor na linguagem de programação JavaScript, Node.js, responsável pela comunicação entre o site, o banco de dados, e o microcontrolador.
- Sistema do hardware: Microcontrolador ESP-32 responsável pelo funcionamento do sistema final, que controla o horário e a quantidade da saída de ração, juntamente com outros componentes como sensor de carga e servo motor.

2.3 Desenvolvimento do sistema

2.3.1 Criação e Implementação dos Arquivos de Código para o ESP32

A programação do ESP32 utilizou a linguagem C++ com o paradigma de Programação Orientada a Objetos (POO). As classes foram organizadas de forma modular para garantir a escalabilidade e facilitar a manutenção do sistema. No apêndice C encontra-se o diagrama de classes.

Classe SensorCarga: responsável por gerenciar o sensor de carga, foi implementada para realizar a leitura do peso e permitir a calibração.

Classe MotorController: dedicada ao controle do servo motor, incluindo os métodos para abrir e fechar o dispenser.

Classe Refeicao: unificou a lógica do sistema, coordenando a leitura de peso e o acionamento do motor com base nos dados de refeição.

Classe WebSocketConnection: integrou a comunicação entre o ESP32 e o servidor Node.js, processando os comandos recebidos e retornando informações.

2.3.2 Desenvolvimento do Servidor Node.js

Foi desenvolvido um servidor utilizando a linguagem JavaScript com o ambiente Node.js. Este servidor teve o papel de intermediar a comunicação entre o front-end, o banco de dados Firebase e o ESP32. As rotas HTTP foram configuradas para operações como cadastro de usuários, login e registro de refeições. Além disso, o WebSocket foi configurado para troca de comandos em tempo real com o microcontrolador.

2.3.3 Construção da Plataforma Web

A interface web foi criada com HTML, CSS e JavaScript, visando uma interação amigável e intuitiva para o usuário. As páginas foram projetadas para realizar as seguintes funções:

Página Sign-Up: permite o cadastro de novos usuários, enviando os dados ao servidor para armazenamento no banco de dados Firebase.

Página Login: autentica o usuário e redireciona para o dashboard.

Dashboard: oferece opções para cadastrar, listar e gerenciar refeições.

Página Dispenser: conecta-se ao ESP32 via WebSocket, permitindo o controle do dispenser, leitura de peso e calibração do sensor.

2.3.4 Integração com o Firebase

O banco de dados Firebase foi utilizado para armazenar as informações de usuários e refeições. A comunicação entre o servidor e o banco foi implementada utilizando a biblioteca firebase-admin. As regras de leitura e escrita foram configuradas para garantir a segurança e integridade dos dados.

2.3.5 Montagem do Circuito Eletrônico

O circuito foi montado utilizando o ESP32, o servo motor e o sensor de carga. Estes componentes foram conectados de forma a garantir uma comunicação eficiente e precisa entre o hardware e o software embarcado.

2.3.6 Testes e Validações

O sistema foi testado para verificar a integração dos componentes. Foram realizados testes de conexão Wi-Fi, comunicação via WebSocket, leitura de peso pelo sensor, funcionamento do servo motor, e envio e recepção de dados pelo servidor.

3 Fundamentação teórica

3.1 Plataformas e linguagens

Diversos artifícios, plataformas e linguagens foram usados durante os meses de produção desse trabalho. Entre elas estão: Firebase, Wokwi, Visual Studio Code, HTML, CSS, Node.js e C + +.

3.1.1 Firebase - Google

O Firebase é uma plataforma de desenvolvimento multiplataforma criada pelo Google. Pode-se pensar nela como uma caixa de ferramentas repleta de recursos para melhorar e expandir aplicativos de maneira mais eficiente. A ferramenta utilizada no projeto foi o *Realtime Database*, um banco de dados salvo na nuvem, sendo possível acessá-lo remotamente através do servidor ou conectá-lo a uma aplicação já existente. A plataforma tem dois planos, um de uso totalmente gratuito e outro de uso pago, que vai acrescendo valor conforme as funcionalidades utilizadas, como hospedagem de apps e autenticação. Para a finalidade desse projeto, o de uso gratuito foi suficiente.

3.1.2 Wokwi

Wokwi é um simulador de eletrônica online. Pode ser usado para simular Arduino, ESP-32 e muitos outros microcontroladores, componentes e sensores populares. Foi utilizado durante o projeto para os testes e simulações envolvendo o funcionamento do circuito, os componentes de hardware e a programação em C + +.

3.1.3 Visual Studio Code

O Visual Studio Code é um editor de código-fonte desenvolvido pela Microsoft. Ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código. Ele é customizável, permitindo que os usuários possam

mudar o tema do editor, teclas de atalho e preferências. Ele é um software livre e de código aberto, e foi a principal ferramenta utilizada para criar e compilar os códigos em HTML e Javascript.

3.1.4 HTML (Linguagem de Marcação de HiperTexto)

HTML é o bloco de construção mais básico da web. Define o significado e a estrutura do conteúdo da web. "Hipertexto" refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. O site é feito com base nessa tecnologia e também utiliza CSS para estilização das páginas web.

3.1.5 CSS (Folhas de Estilo em Cascata)

CSS3 é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML. Descreve como elementos são mostrados na tela, no papel, na fala ou em outras mídias. É uma das principais linguagens da open web e é padronizada em navegadores web de acordo com as especificações da W3C. Foi utilizada no projeto para estilização e personalização das páginas HTML.

3.1.6 Node.js

O Node.js é um ambiente de execução do código JavaScript do lado servidor (server side), que na prática se reflete na possibilidade de criar aplicações standalone (autossuficientes) em uma máquina servidora, sem a necessidade do navegador. O Node.js foi utilizado no projeto para comunicação do front-end (HTML) com o ESP-32, hospedando o site localmente e enviando comandos para o microcontrolador com base na interação do usuário.

3.1.7 C + +

C + + é uma linguagem de programação de nível médio, baseada na linguagem C. É a linguagem utilizada nos microcontroladores Arduino e ESP-32, e

nesse projeto ela é utilizada juntamente com o paradigma de programação orientada a objetos (poo), para controle dos componentes do circuito e do próprio ESP32.

3.2 Paradigma de Programação Orientada a Objetos (POO)

O paradigma de Programação Orientada a Objetos (POO) foi o alicerce para o desenvolvimento do software embarcado no ESP32. A POO organiza o código em classes, que encapsulam atributos e métodos, promovendo a reutilização, modularidade e manutenção do código. As principais características do paradigma aplicadas no projeto foram:

Encapsulamento: cada componente do sistema, como o motor, sensor de carga e a lógica de refeição, foi encapsulado em classes específicas.

Abstração: a complexidade interna de cada componente foi ocultada, permitindo que o restante do sistema interagisse com eles de forma simplificada.

Herança e Polimorfismo: essas características não foram utilizadas diretamente, mas a estrutura modular permite fácil implementação futura caso sejam necessárias.

4 Resultados e discussões

4.1 Telas Desenvolvidas

As páginas do sistema foram criadas para oferecer uma interface amigável e funcional. Abaixo estão os principais resultados.

Página de apresentação: apresenta de forma curta e objetiva o projeto e direciona o usuário para o login ou cadastro (figura 3).

Figura 3 - Tela landing page

Página de Cadastro (Sign-Up): permite o registro de novos usuários (figura 4).

Figura 4 - Tela sign-up

Página de Login: autentica os usuários cadastrados (figura 5).

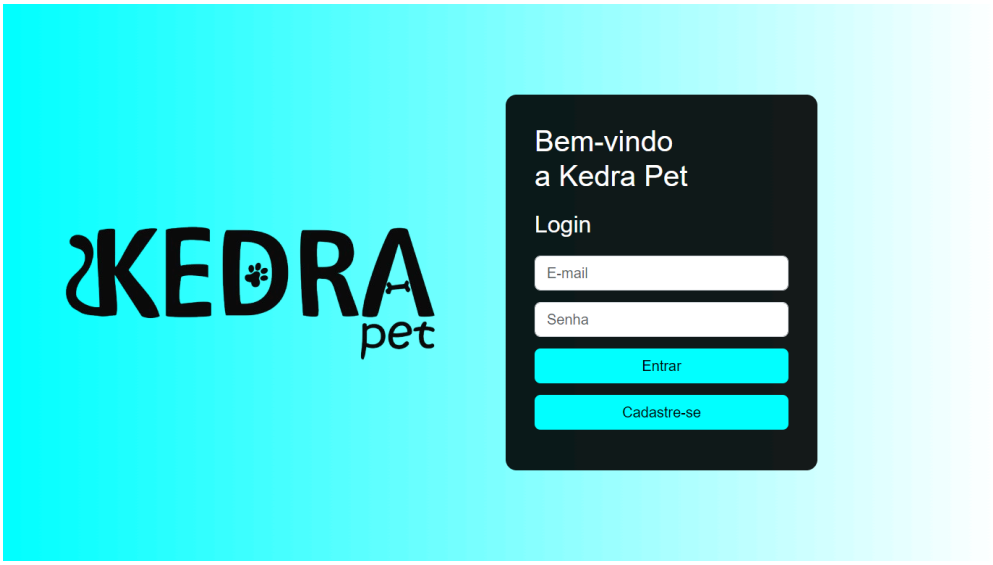


Figura 5 - Tela de login

Dashboard: gerencia refeições e oferece um botão de acesso ao controle do dispenser. Ao clicar em Ver Refeições, aparece uma lista das refeições criadas pelo usuário (figura 6).

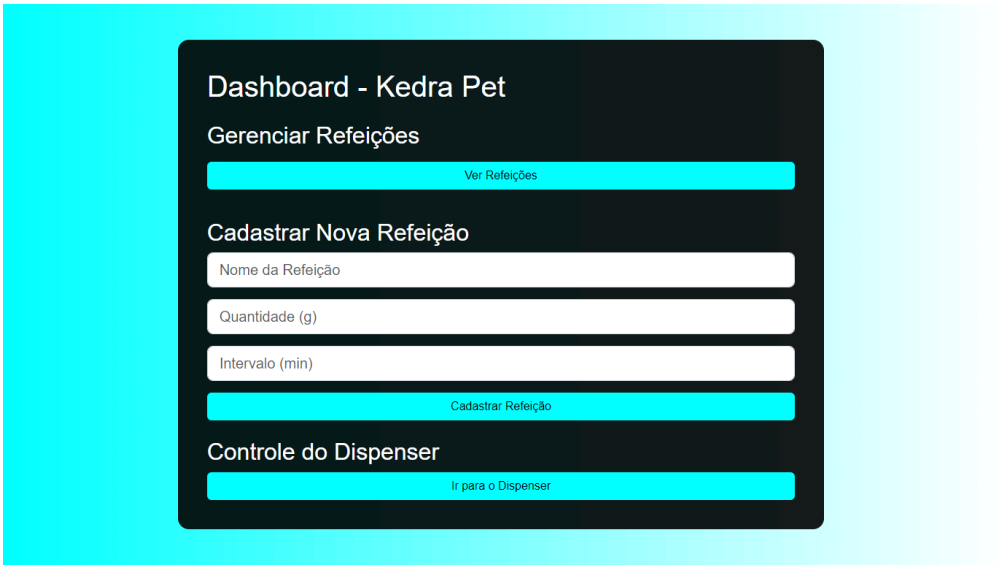


Figura 6 - Tela dashboard

Página Dispenser: conecta-se ao ESP-32 e permite comandos como calibrar balança, distribuir alimento e ler peso atual (figura 7).

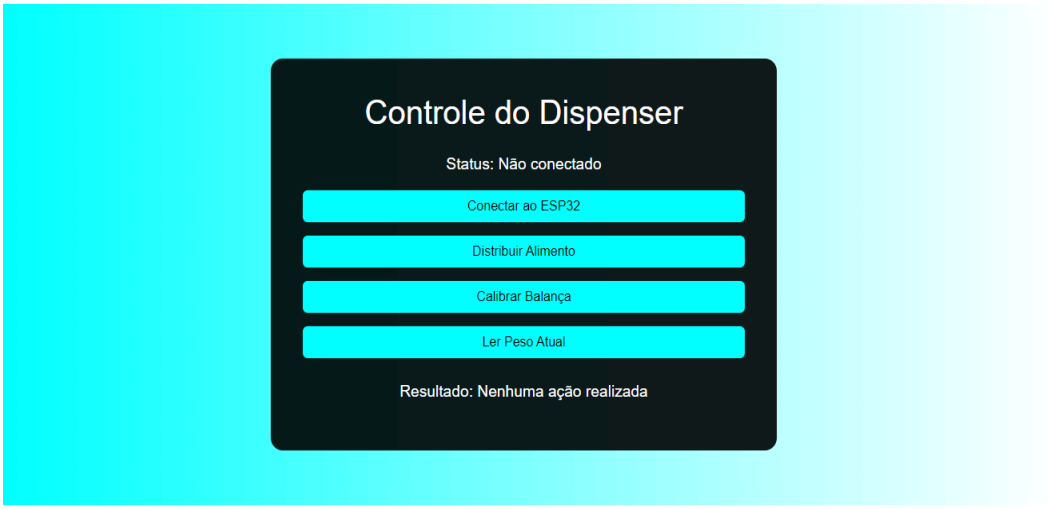


Figura 7 - Tela dispenser

4.2 Criação e Implementação dos Arquivos de Código para o ESP32

A programação do ESP32 utilizou a linguagem C++ com o paradigma de Programação Orientada a Objetos (POO). As classes foram organizadas de forma modular para garantir a escalabilidade e facilitar a manutenção do sistema, tendo dois arquivos cada classe, um .h e outro .cpp.

Arquivos .h (Cabeçalhos):

Os arquivos de cabeçalho (.h) definem as classes, atributos e métodos utilizados no projeto. Eles encapsulam a lógica e garantem a modularidade do código.

MotorController.h

Função: Controlar o servo motor responsável pela abertura e fechamento do dispenser.

Atributo:

pino: pino GPIO conectado ao servo motor.

Métodos:

iniciar: Inicializa o servo motor.

abrir: Move o servo motor para abrir o dispenser.

fechar: Retorna o servo motor à posição inicial.

código MotorController.h:

```
#ifndef MOTORCONTROLLER_H
#define MOTORCONTROLLER_H

#include <Servo.h>

class MotorController {
private:
    Servo motor;
    int pin;
    int anguloAberto;
    int anguloFechado;

public:
        MotorController(int pin, int anguloAberto = 90, int
anguloFechado = 0);
    void iniciar();
    void abrir();
    void fechar();
};

#endif
```

SensorCarga.h

Função: Gerenciar o sensor de carga e realizar leituras de peso.

Atributos - pinoDout e pinoSck: pinos GPIO conectados ao módulo HX711.

fatorCalibracao: define o fator de calibração do sensor.

Métodos:

iniciar: Configura o módulo HX711.

lerPeso: Retorna o peso atual lido pelo sensor.

código SensorCarga.h:

```
#ifndef SENSORCARGA_H
#define SENSORCARGA_H

#include <Arduino.h>
#include <HX711.h>

class SensorCarga {
private:
    HX711 balanca;
    int pinDOUT;
    int pinSCK;
    float fatorCalibracao;

public:
    SensorCarga(int pinDOUT, int pinSCK, float fatorCalibracao);
    void iniciar();
    float lerPeso();
    void calibrar(float fatorCalibracao);
};
```

Refeicao.h

Função: Coordenar a lógica de distribuição de alimentos.

Atributos - motor: instância de MotorController.

sensor: instância de SensorCarga.

Métodos:

configurarBalanca: Configura o sensor para calibrar e zerar o peso.

distribuirAlimento: Gerencia a distribuição com base na quantidade e peso.

código Refeicao.h:

```
#ifndef REFEICAO_H
#define REFEICAO_H

#include <Arduino.h>
#include "MotorController.h"
#include "SensorCarga.h"

class Refeicao {
private:
    MotorController* motor;
    SensorCarga* sensor;

public:
    Refeicao(MotorController* motor, SensorCarga* sensor);
    void configurarBalanca();
    float obterPesoAtual();
    void distribuirAlimento(float quantidade);
};
```

WebSocketConnection.h

Função: Estabelecer a comunicação entre o ESP32 e o servidor Node.js via WebSocket.

Atributos - webSocket: objeto para gerenciar a conexão.

Métodos:

conectar: Estabelece a conexão WebSocket.

handleWebSocketEvent: Lida com eventos e comandos recebidos do servidor.

código WebSocketConnection.h

```
#ifndef WEBSOCKETCONNECTION_H
#define WEBSOCKETCONNECTION_H

#include <WebSocketsClient.h>
#include <ArduinoJson.h>
#include "Refeicao.h"
#include "MotorController.h"
#include "SensorCarga.h"

class WebSocketConnection {
private:
    WebSocketsClient webSocket;
    Refeicao* refeicao;

    // Método privado para lidar com eventos WebSocket
       void handleWebSocketEvent(WStype_t type, uint8_t* payload,
size_t length);
```

```
public:
    // Construtor
       WebSocketConnection(const char* host, int port, Refeicao*
refeicao);

    // Métodos públicos
    void conectar();
    void loop();
};

#endif
```

Arquivos .cpp (Implementação):

Os arquivos .cpp implementam as funcionalidades declaradas nos arquivos .h.

MotorController.cpp: Implementa os métodos para controlar o servo motor.

```
#include "MotorController.h"


MotorController::MotorController(int pin, int anguloAberto, int
anguloFechado)
                    :     pin(pin),     anguloAberto(anguloAberto),
anguloFechado(anguloFechado) {}


void MotorController::iniciar() {
    motor.attach(pin);
    fechar();  // Fecha o motor na inicialização
    Serial.println("Motor iniciado e fechado.");
}


void MotorController::abrir() {
    motor.write(anguloAberto);
     delay(1000);   // Aguarda 1 segundo para garantir a abertura
completa
    Serial.println("Motor aberto.");
}


void MotorController::fechar() {
    motor.write(anguloFechado);
     delay(1000);   // Aguarda 1 segundo para garantir o fechamento
completo
    Serial.println("Motor fechado.");
```

SensorCarga.cpp: Gerencia as leituras de peso do sensor.

```
#include "SensorCarga.h"


SensorCarga::SensorCarga(int pinDOUT, int pinSCK, float
fatorCalibracao)
                        :     pinDOUT(pinDOUT),     pinSCK(pinSCK),
fatorCalibracao(fatorCalibracao) {}


void SensorCarga::iniciar() {
    balanca.begin(pinDOUT, pinSCK);
    balanca.set_scale(fatorCalibracao);
    balanca.tare();  // Reseta o peso inicial para 0
    Serial.println("Sensor de carga iniciado.");
}


float SensorCarga::lerPeso() {
    if (balanca.is_ready()) {
         float peso = balanca.get_units(10); // Retorna a média de
10 leituras
         return peso > 0 ? peso : 0.0;       // Garante que o peso
não seja negativo
    } else {
        Serial.println("Sensor de carga não está pronto!");
        return -1.0;  // Indica erro
    }
}


void SensorCarga::calibrar(float fatorCalibracao) {
    this->fatorCalibracao = fatorCalibracao;
```

```
    balanca.set_scale(fatorCalibracao);
    Serial.println("Balança calibrada.");
}
```

Refeicao.cpp: Implementa a lógica para distribuir alimento com base no peso lido pelo sensor.

```
#include "Refeicao.h"

Refeicao::Refeicao(MotorController* motor, SensorCarga* sensor)
    : motor(motor), sensor(sensor) {}

void Refeicao::configurarBalanca() {
    Serial.println("Configurando balança...");
    sensor->iniciar();
    sensor->calibrar(1100.0); // Fator de calibração ajustável
    Serial.println("Balança configurada.");
}

float Refeicao::obterPesoAtual() {
    float pesoAtual = sensor->lerPeso();
    Serial.println("Peso atual lido: " + String(pesoAtual) + "g");
    return pesoAtual;
}

void Refeicao::distribuirAlimento(float quantidade) {
    Serial.println("Iniciando distribuição de alimento...");
    motor->abrir();
    while (sensor->lerPeso() < quantidade) {
```

```
        delay(500);  // Aguarda meio segundo para verificar o peso
novamente
    }
    motor->fechar();
    Serial.println("Distribuição concluída!");
}
```

WebSocketConnection.cpp: Implementa a lógica para gerenciar comandos recebidos via WebSocket.

```
#include "WebSocketConnection.h"

WebSocketConnection::WebSocketConnection(const char* host, int
port, Refeicao* refeicao)
    : refeicao(refeicao) {
    webSocket.begin(host, port, "/");
        webSocket.onEvent([this](WStype_t type, uint8_t* payload,
size_t length) {
        this->handleWebSocketEvent(type, payload, length);
    });
}

void WebSocketConnection::conectar() {
    webSocket.setReconnectInterval(5000);
    webSocket.sendTXT("esp32-client");
    Serial.println("Conexão WebSocket estabelecida.");
}

void    WebSocketConnection::handleWebSocketEvent(WStype_t    type,
uint8_t* payload, size_t length) {
```

```
    if (type == WStype_TEXT) {
        DynamicJsonDocument doc(1024);
        deserializeJson(doc, payload);

        String acao = doc["acao"];

        // Evento: Distribuir Alimento
        if (acao == "distribuirAlimento") {
            float quantidade = doc["dados"]["quantidade"];
            refeicao->distribuirAlimento(quantidade);

webSocket.sendTXT("{\"evento\":\"distribuirAlimento\",\"status\":\"
sucesso\"}");

        // Evento: Calibrar Balança
        } else if (acao == "calibrarBalanca") {
            refeicao->configurarBalanca();

webSocket.sendTXT("{\"evento\":\"calibrarBalanca\",\"status\":\"suc
esso\"}");

        // Evento: Ler Peso Atual
        } else if (acao == "lerPesoAtual") {
            float pesoAtual = refeicao->obterPesoAtual();
                                        String resposta =
"{\"evento\":\"lerPesoAtual\",\"peso\":" + String(pesoAtual) + "}";
            webSocket.sendTXT(resposta);

        // Evento: Status do Motor
```

```
        } else if (acao == "statusMotor") {
            bool motorAberto = refeicao->motor->estaAberto();
                                                String resposta =
"{\"evento\":\"statusMotor\",\"aberto\":" + String(motorAberto) +
"}";
            webSocket.sendTXT(resposta);

        } else {
            Serial.println("Ação não reconhecida: " + acao);

webSocket.sendTXT("{\"evento\":\"erro\",\"mensagem\":\"Ação não
reconhecida\"}");
        }
    }
}

void WebSocketConnection::loop() {
    webSocket.loop();
}
```

***Arquivo Main.cpp*:** integra todas as classes e define a lógica principal de funcionamento do sistema.

Conexão Wi-Fi: Configura a conexão à rede para comunicação com o servidor.

Inicialização do Hardware: Configura o servo motor e o sensor de carga.

Loop Principal: Mantém a conexão WebSocket ativa e aguarda comandos.

```
#include <WiFi.h>
#include "WebSocketConnection.h"
#include "Refeicao.h"
#include "MotorController.h"
```

```
#include "SensorCarga.h"

// Configurações de rede e servidor
const char* ssid = "SEU_SSID";           // Substitua pelo nome da sua rede Wi-Fi
const char* password = "SUA_SENHA";     // Substitua pela senha da sua rede Wi-Fi
const char* serverHost = "192.168.1.100"; // IP do servidor WebSocket
const int serverPort = 8080;                // Porta do servidor WebSocket

// Objetos globais
MotorController* motor;
SensorCarga* sensor;
Refeicao* refeicao;
WebSocketConnection* webSocketConnection;

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

    // Conecta ao Wi-Fi
    WiFi.begin(ssid, password);
    Serial.print("Conectando ao Wi-Fi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
```

```
    }
    Serial.println("\nConectado ao Wi-Fi!");

    // Inicialização dos componentes
    motor = new MotorController(5); // Substitua pelo pino do servo motor
     sensor = new SensorCarga(2, 4, 1100.0); // Pinos DOUT, SCK e fator de calibração
    refeicao = new Refeicao(motor, sensor);

    // Inicializa o motor e configura a balança
    motor->iniciar();
    refeicao->configurarBalanca();

    // Inicializa o WebSocket
        webSocketConnection = new WebSocketConnection(serverHost, serverPort, refeicao);
    webSocketConnection->conectar();

    Serial.println("Sistema inicializado.");
}

void loop() {
    // Mantém a conexão WebSocket ativa
    webSocketConnection->loop();

    // Pequena pausa para evitar sobrecarga
    delay(10);
}
```

4.3 Servidor Node.js

O servidor Node.js atua como intermediário entre o front-end e o ESP32, além de gerenciar a persistência de dados no Firebase. Ele utiliza HTTP para operações CRUD e WebSocket para comunicação em tempo real. Segue estrutura da pasta:

```
kedra-server/
├── server.js  # Arquivo principal do servidor
├── package.json     # Configurações do projeto Node.js
├── package-lock.json     # Versão bloqueada das dependências instaladas
├── public/   # Arquivos do front-end
│   ├── login.html
│   ├── sign-up.html
│  ├── dashboard.html
│   ├── dispenser.html
│   ├── style.css
    ── serviceAccountKey.json     # Chave de autenticação do Firebase (arquivo privado)
├── utils/
```

4.4 Construção do circuito

A construção do circuito no ESP-32 é um dos elementos centrais para o funcionamento do sistema. O circuito combina os componentes de hardware (ESP-32, servo motor e sensor de carga) para executar as tarefas de automação da alimentação de pets. Abaixo, apresento uma descrição dos componentes e uma imagem do circuito, exposto na figura 8.

ESP-32: Microcontrolador central do sistema, que gerencia a lógica e realiza a comunicação Wi-Fi com o servidor. Permite conectar sensores e atuadores, oferecendo GPIOs para entrada e saída.

Servo Motor: Controla a abertura e o fechamento do dispenser de alimentos. Conectado a um pino PWM do ESP32 para controle do movimento.

Sensor de Carga (com Módulo HX711): Mede o peso dos alimentos despejados, garantindo que a quantidade programada seja distribuída.

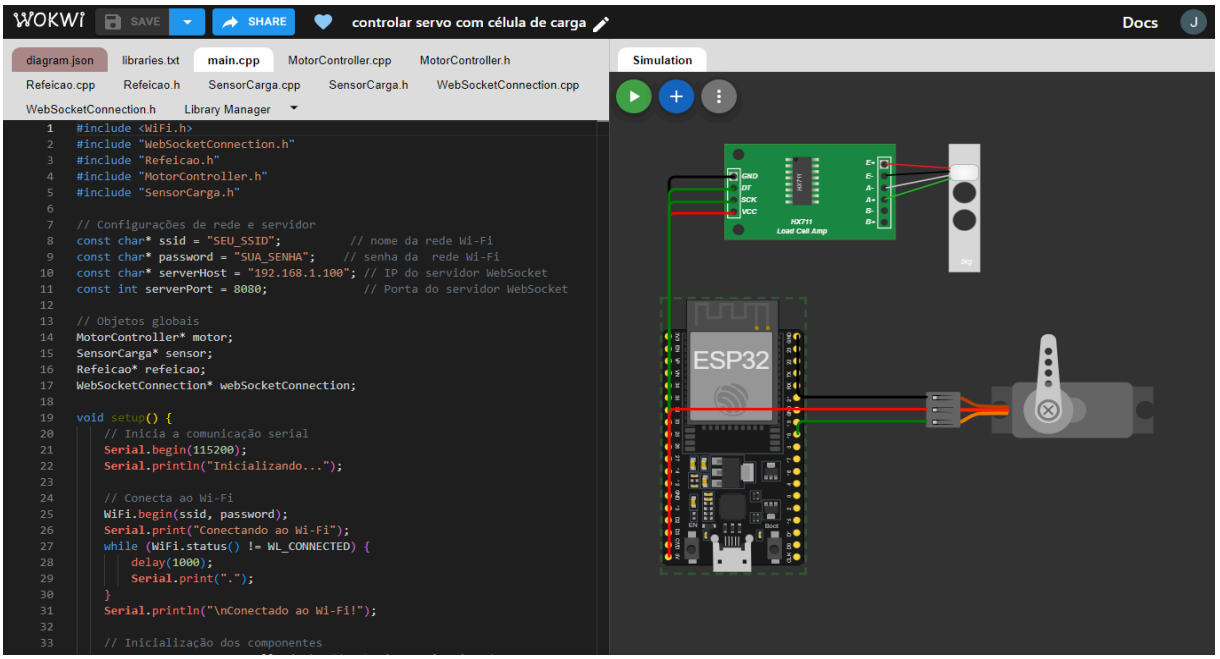


Figura 8 - Circuito digital

5 Conclusões

5.1 Objetivos Alcançados

O uso do servo motor, controlado pelo ESP32, permitiu a distribuição de alimentos com precisão, baseada em uma quantidade definida pelo usuário. O sensor de carga, juntamente com o módulo HX711, possibilitou a leitura do peso dos alimentos, garantindo que a quantidade programada fosse atingida antes de fechar o dispenser. O sistema foi projetado com uma estrutura modular, onde cada componente de hardware (sensor de carga e servo motor) foi controlado por classes específicas, utilizando o paradigma de Programação Orientada a Objetos (POO). Isso garantiu uma arquitetura escalável e de fácil manutenção. O servidor Node.js, que funciona como intermediário entre o front-end e o ESP32, foi configurado para gerenciar as requisições de cadastro de usuários, login, e comandos enviados para o ESP32, além de garantir a persistência dos dados no banco de dados Firebase. A interface do usuário foi construída com HTML e CSS oferecendo um painel intuitivo para o controle das refeições e do dispenser de alimentos. O design foi mantido simples, mas funcional, permitindo ao usuário interagir facilmente com o sistema. A comunicação entre o servidor e o ESP32 foi realizada utilizando WebSocket, permitindo que os comandos fossem enviados em tempo real, sem necessidade de atualizações contínuas, proporcionando uma experiência de usuário mais fluida e eficiente.

5.2 Possíveis melhorias futuras

Algumas melhorias podem ser implementadas no futuro, como: adição de notificações, melhorias na interface, controle via aplicativo móvel, integração com sensores adicionais.

Este projeto demonstrou a viabilidade de criar um sistema de automação para alimentar pets utilizando o ESP-32, servo motor, sensor de carga e integração com uma plataforma web. O uso de tecnologias como WebSocket, Firebase e Node.js garantiu que a solução fosse eficiente e escalável, permitindo um controle completo sobre o processo de alimentação. O sucesso do projeto pode servir como base para

o desenvolvimento de sistemas mais complexos e personalizados, voltados para a automação doméstica e cuidados com animais de estimação.

6 Referências

MEDEIROS, Débora; LARISSA, Stephanie. **Crescimento do mercado pet e oportunidade de negócio.** [S. l.], 12 ago. 2024. Disponível em: https://sebrae.com.br/sites/PortalSebrae/ufs/al/artigos/crescimento-do-mercado-pet-e-oportunidade-de-negocio,021731b7fe057810VgnVCM1000001b00320aRCRD. Acesso em: 14 ago. 2024.

SILVA, Daniel Neves. **História da internet.** Brasil Escola. Disponível em: https://brasilescola.uol.com.br/informatica/internet.htm. Acesso em 14 de agosto de 2024.

MOZILLA. MDN Web Docs. HTML. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 08 dez. 2024.

WOKWI. Wokwi - Simulador de Circuitos. Disponível em: https://wokwi.com. Acesso em: 08 dez. 2024.

MOZILLA. MDN Web Docs. CSS. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/CSS. Acesso em: 08 dez. 2024.

ALURA. Node.js. Disponível em: https://www.alura.com.br/artigos/node-js?srsltid=AfmBOooG_OvECvGtg_8KohYX_5nEQsKCOAvnRMye3kiANdAfv5DInm5V. Acesso em: 08 dez. 2024.

PORTAL G1. Uso da internet no Brasil cresce e 70% da população está conectada. *G1 - Economia e Tecnologia*, 28 ago. 2019. Disponível em: https://g1.globo.com/economia/tecnologia/noticia/2019/08/28/uso-da-internet-no-brasil-cresce-e-70percent-da-populacao-esta-conectada.ghtml. Acesso em: 16 dez. 2024.

SEBRAE. Crescimento do mercado pet é oportunidade de negócio. *Portal Sebrae*, [s.d.]. Disponível em: https://sebrae.com.br/sites/PortalSebrae/ufs/al/artigos/crescimento-do-mercado-pet-e-oportunidade-de-negocio,021731b7fe057810VgnVCM1000001b00320aRCRD. Acesso em: 16 dez. 2024.

Link do projeto no GitHub: https://github.com/JuanGabrielGomes/Kedra-Pet.git
Link do projeto no Wokwi: https://wokwi.com/projects/413043738404595713

Apêndice A - Documento de requisitos

PROBLEMA DE NEGÓCIO

Controlar um dispenser de ração para animais de estimação (pequeno e médio porte), sendo possível definir os horários de liberação do alimento e a quantidade (em gramas) de cada alimentação.

REQUISITOS FUNCIONAIS

RF1 – CADASTRAR OU LOGAR USUÁRIO

Ao entrar na plataforma pela primeira vez, será necessário se cadastrar com nome, e-mail e senha. Após isso, o usuário será automaticamente direcionado para o login e terá acesso ao painel principal do site.

RF2 – CADASTRAR ALIMENTAÇÃO

Essa função será primordial. Nela, o usuário irá cadastrar todas as refeições do seu animal de estimação, especificando nome da alimentação, quantidade em gramas e horário de recorrência, ou seja, em quantos minutos a refeição será liberada.

RF3 – VISUALIZAR REFEIÇÕES

Nessa aba, o usuário irá visualizar as alimentações cadastradas e suas especificações, entre elas seus respectivos horários, nomes e quantidades.

RF4 – CONECTAR DISPENSER

Nessa área, o usuário irá se conectar com um dispenser que esteja na mesma rede local através da conexão WebSocket do ESP-32, gerenciada pelo servidor Node.js.

RF4.1 – DISTRIBUIR ALIMENTO INSTANTANEAMENTE

Uma vez conectado ao dispenser, o usuário terá acesso a mais algumas funções adicionais relacionadas ao circuito físico, como “distribuir alimento”, que ao clicar, o dispenser liberará automaticamente uma quantidade de ração definida pelo usuário.

RF4.2 - CALIBRAR BALANÇA

É uma funcionalidade adicional, assim como “distribuir”, que irá calibrar a balança automaticamente.

RF4.3 - LER PESO

Faz uma leitura do sensor de carga do projeto, para saber se ainda resta alguma ração no pote onde ela foi despejada.

REQUISITOS NÃO FUNCIONAIS

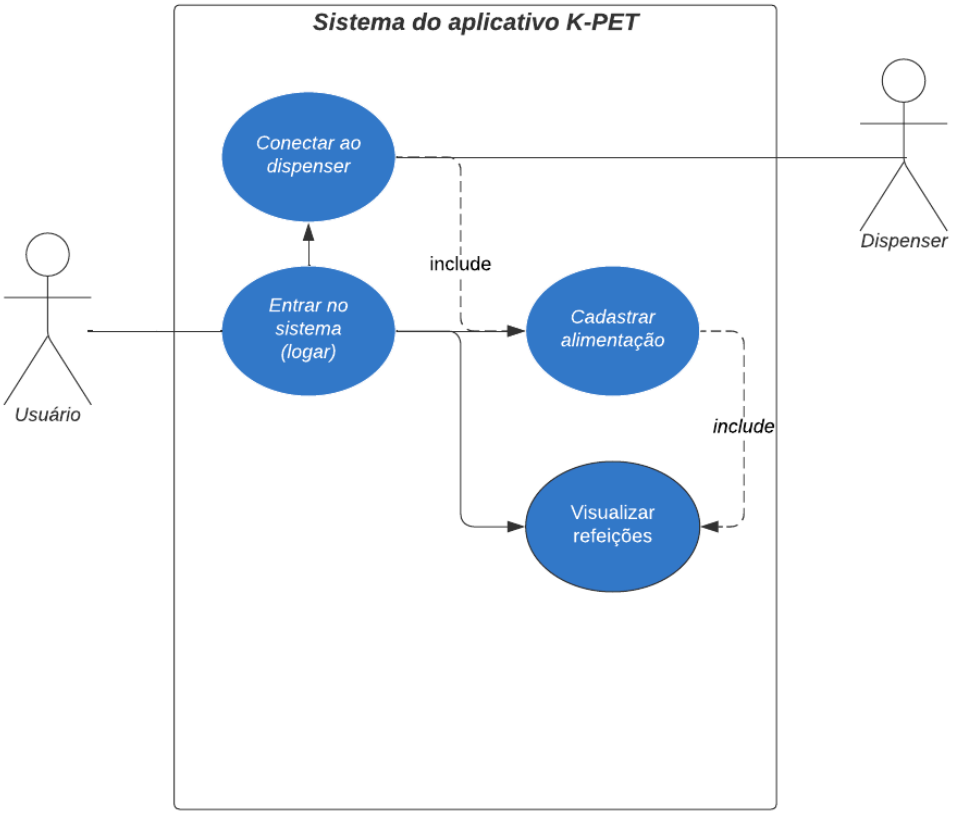
RNF1 - PLATAFORMA WEB

O sistema criado deve ser desenvolvido para web, sendo acessado por qualquer navegador disponível.

RNF2 - PROGRAMAÇÃO ORIENTADA A OBJETO

O desenvolvimento do sistema deverá seguir o paradigma de programação baseada no conceito de objetos, atributos e métodos.

Apêndice B - Diagrama de casos de uso



No sistema, o usuário deverá conectar-se através do seu login, tendo assim acesso ao painel principal. No painel, o usuário poderá cadastrar uma nova refeição, com nome, quantidade em gramas e horário. Além de visualizar as refeições já criadas e acessar o dispenser, há outras opções adicionais como calibrar balança e distribuir alimento, já incluídas na página de dispenser.

Apêndice C - Diagrama de classes