**UNIVERSIDADE DE MARÍLIA (UNIMAR)**

**Bacharelado em Ciências da Computação**

Programação Orientada a Objetos

**Sistema Integrado de Cuidado e Monitoramento de Pets**

**Nome:** José Victor Santos Tranquilino de Souza

**RA**: 1963245

**Descrição do Projeto**

Esse projeto tem como objetivo o monitoramento das atividades do animal de estimação através de uma coleira com dados da localização e saúde do animal. Tendo também um comedouro inteligente com possibilidade de agendamento dos horários da alimentação. Tudo isso pode ser monitorado através de um aparelho móvel celular.

O projeto utiliza o termo IOT(Internet of Things), que mistura aparelhos físicos, como o microcontrolador ESP-32, junto ao uso de softwares, sendo esse o banco de dados. O banco de dados utilizado neste projeto é o Firebase, que funciona em tempo real e é eficiente para a integração no Arduino. Este projeto foi desenvolvido na linguagem C++ na IDE do Arduino.

**Requisitos**

**Conceitos Básicos:**

**“Codificou classe - Quais classes? Para que servem?”**

Há duas classes codificadas. A “ConnectionWifi” e “ConnectionFirebase”.

**ConnectionWifi:** Essa classe tem como funcionalidade definir os parâmetros de conexão com uma rede Wi-Fi.

**ConnectionFirebase:** Essa classe serve para definir os parâmetros de conexão com o Firebase.

**“Codificou atributos - Em quais classes? Quais atributos? Por que esses nomes e tipos?”**

Foram utilizados diversos atributos em ambas as classes.

**ConnectionWifi:** Os atributos são “**char\* ssid**”, responsável por armazenar o nome da rede Wi-Fi, e “**char\* password**”, responsável por armazenar a senha da respectiva rede Wi-Fi. Ambos são do tipo **char**, pois pode haver letras e números no nome e na senha.

**ConnectionFirebase:** Os atributos são “**char\* FIREBASE\_HOST**”,responsável por armazenar o servidor ou URL da base de dados do Firebase, e “**char\* FIREBASE\_AUTH**”, responsável por armazenar o token de conexão com o Firebasa. Ambos em tipo de **char**, pois podem conter algarismos numéricos e não numéricos.

Todos os atributos são ponteiros pois apontam para o primeiro caractere da sequência, economizando memória.

“**Codificou atributos estáticos - Em quais classes? Por que eles são estáticos? Por que esses nomes e tipos?**”

Todos os atributos utilizados nas classes são estáticos, porque as informações que representam são de constante acesso e não devem ser alteradas para não ocorrer erro na conexão do ESP-32 com o Firebase. Os nomes são de fácil dedução para seus propósitos de armazenar usuários e conexões.

“**Codificou métodos - Em quais classes? Quais métodos? Por que o esses nomes e parâmetros?**”

Diversos métodos foram utilizados nas classes.

**Refeição:**

**->getHorario():** Retorna o horário da refeição que o usuário definir.

**->setHorario(const Horario& novoHorario):** Defini um novo horário de alimentação do pet a partir do horário que o pet come.

**->getRefeicaoFeita():** Retorna se a refeição foi feita ou não.

**->ehHorarioReifeicao(const Horario& agora):** Compara o horário atual com o horário da refeição do pet e verifica se é hora da reifeição ou não.

“**Codificou métodos construtores - Em quais classes?**”

Nas classes “ConnectionWifi” e “ConnectionFirebase” métodos construtores para definir os valores dos atributos.

**ConnectionWifi Código:**

ConnectionWifi() : ssid(“esp32”), password(“12345678”) {}

**ConnectionFirebase Código:**

ConnectionFirebase() : FIREBASE\_HOST(“<https://linserver-default-rtdb.firebaseio.com/>”), FIREBASE\_AUTH(“Token”) {}

“**Codificou atributos protegidos e/ou privados - Em quais classes? Quais atributos? Por que não são públicos?**”

Há diversas classes com atributos privados.

**Classe Horário:**

**-> int hora;**

**-> int minuto;**

**Classe Refeicao:**

**-> Horario horarioDefinido;**

**-> bool refeicaoFeita;**

Todos estes atributos são privados, pois não devem ser alterados por meios externos, diferente dos da classe ConnectionWifi que servem para se conectar com a rede Wi-Fi que o usuário inseriu os valores.

“**Instanciou objetos - Quais objetos?**”

ConnectionWifi conexaoWi-Fi.

ConnectionFirebase conexaoFirebase;

“**Instalou e usou bibliotecas de terceiros - Quais bibliotecas? Para que servem?**”

**->** **#include <Firebase\_ESP\_Client.h>:** Conecta o ESP-32 com a Firebase, mas depende da biblioteca FirebaseEsp32.h.

**-> #include <PulseSensorPlayground.h>:** Serve paracontrolar o sensor de pulso.

**-> #include <WiFi.h>:** Serve para conectar o ESP-32 ao Wi-Fi.

**-> #include <FirebaseESP32>:** Serve para conectar o ESP-32 com o Firebase.

**-> #include <TinyGPSPlus.h>:** Conecta o GPS com os satélites próximos e lê seus dados do GPS.

**Design**

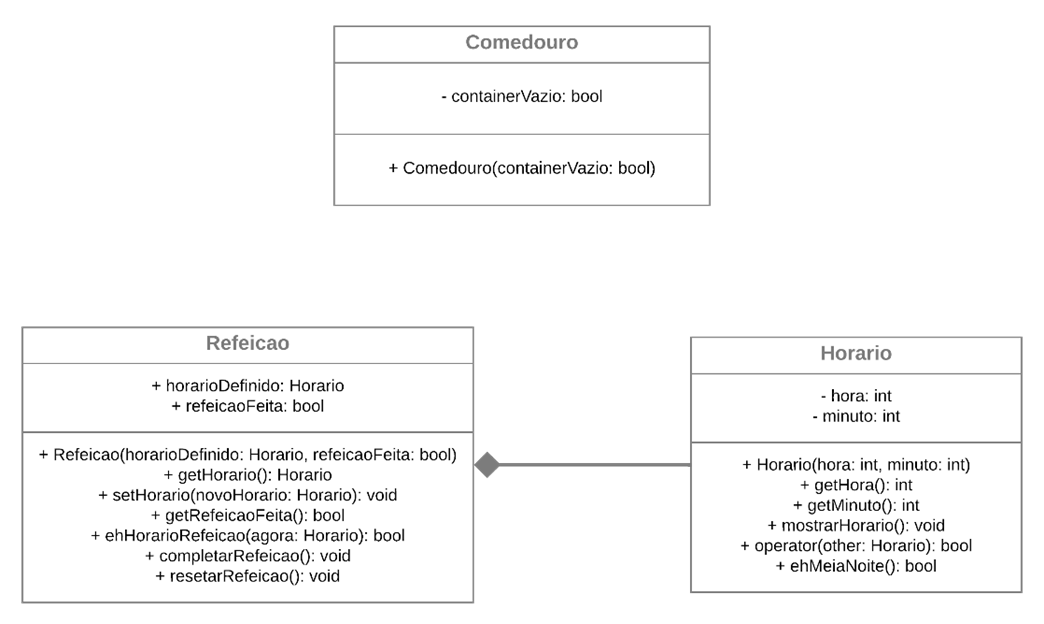
**“Identificou e codificou classes de dados - Quais classes?”**

**-Classe Comedouro;**

**-Classe Horario;**

**-Classe Refeicao;**

**“Usou diagramas UML para discutir a solução - Fazer upload dos diagramas”**

****

**“Ocultou informações usando atributos e ou métodos protected/private”**

Os atributos das classes “Horario” e “Refeicao” são privados para não serem alterados por meios externos e restringir as informações dentro das classes.

**Boas Práticas**

**“Codificou testes unitários - Como executou os testes?”**

Muitos testes foram realizados com diversos recursos como bibliotecas e classes e agora estão funcionando no projeto, além de o uso de sites com Worki e Thinkercad foram úteis na hora dos testes. Adicionando partes a serem testados no código base.

“**Codificou padrões de projeto - Quais padões? Onde?**”

Para a nomeação das classes foi utilizado o padrão PascalCase a partir do código de conexão(connection.ino).

“**Usou conceitos de código limpo/SOLID - Quais conceitos? Onde?**”

A nomeação das classes e variáveis foram de forma descritivas e diretas. Formas de tratamento de erros e identações com espaçamento também foram utilizadas.

O código foi organizado de forma lógica para ser o mais claro possível, com o intenção de não precisar de comentários

**Boas Práticas**

**“Publicou todo o projeto integrador no Gitlab, Github ou semelhantes - Enviar link”**

<https://github.com./leticia-hub/projeto-integrador/tree/main>

**Parte 2 - Atualizações**

**“Identificou e codificou classes de comportamento - Quais classes?”**

A **“class Refeicao”** controla o comedouro com base nos dados da firebase, lendo-os e definindo se será feita ou não a refeição.

class Refeicao {

private:

Horario horarioDefinido;

bool refeicaoFeita;

public:

Refeicao(const Horario & horarioDefinido, bool refeicaoFeita):

horarioDefinido(horarioDefinido), refeicaoFeita(refeicaoFeita) {}

Horario getHorario() const {

return horarioDefinifo;

}

void setHorario(const Horario & novoHorario){  
 horarioDefinido = novoHorario;

}

bool getRefeicaoFeita() const {

return refeicaoFeita;

}

bool ehHorarioRefeicao(const Horario & agora) const {

return agora == horarioDefinido;

}

void completarRefeicao() {

refeicaoFeita = true;

}

void resetarRefeicao() {

refeicaoFeita = false;

}

};

**“Codificou atributos estáticos - Em quais classes? Por que eles são estáticos? Por que esses nomes e tipos?”**

class Comedouro { public: static bool containerCheio;

**“Codificou métodos estáticos”**

static int capacidadeMaxima(){

return 100; // Substitua pelo valor real da capacidade máxima

**“Codificou métodos protegidos e/ou privados”**

private:

void LimitarHora() {

// Por exemplo adicionando 1 hora ao horário

hora = (hora + 1) % 24;

}

**“Usou objetos imutáveis”**

int getHora() const {

return hora;

}

**“Versionou todo o projeto integrador com Git”**

<https://github.com/leticia-hub/projeto-integrador/tree/comedouro-2.0>

<https://github.com/leticia-hub/projeto-integrador/tree/comedouro-3.0>