UNIVERSIDADE FEDERAL DE SÃO CARLOS
Centro de Ciências Exatas e de Tecnologia– CCET Departamento de Computação– DC Trabalho de Conclusão de Curso em Engenharia de Computação

Wanderson Moreira do Nascimento

Criação de Blocos para conexão com banco de dados MySQL no Block-Based Integrated Platform for Embedded Systems (BIPES)

São Carlos - SP

2023

Wanderson Moreira do Nascimento

Criação de Blocos para conexão com banco de dados MySQL no Block-Based Integrated Platform for Embedded Systems (BIPES)

Trabalho de Conclusão de Curso submetido à Universidade Federal de São Carlos, como requisito necessário para obtenção do grau de Bacharel em Engenharia de Computação.

Orientação Prof. Dr. Rafael Vidal Aroca
Coorientação Prof. Dra. Marilde Terezinha Prado Santos

São Carlos - SP

2023

Este trabalho é dedicado a todos aqueles que me ajudaram a chegar até aqui. À minha família, que sempre me apoiou e me deu forças para continuar mesmo nos momentos mais difíceis. Aos amigos, que estiveram ao meu lado e me motivaram a seguir em frente. E aos professores e orientadores, que me guiaram, ensinaram e inspiraram, tornando possível a realização deste sonho.

"Toda a obra de um homem, seja em literatura, música, pintura, arquitetura ou em qualquer outra coisa, é sempre um auto-retrato; e quanto mais ele se tentar esconder, mais o seu caráter se revelará, contra a sua vontade.

(Samuel Butler)

Resumo

A tendência do desenvolvimento sem código, também conhecido como no-code ou low-code, tem ganhado popularidade pela sua capacidade de acelerar o processo de desenvolvimento e permitir que pessoas sem experiência em programação criem soluções de maneira simplificada. Uma solução que se encaixa nessa tendência é o Block based Integrated Platform for Embedded Systems (BIPES), uma plataforma de desenvolvimento de sistemas embarcados baseada em blocos, projetada para tornar mais fácil a criação de projetos eletrônicos sem a necessidade de conhecimento avançado em programação. Com o BIPES, os usuários podem criar programas para controlar dispositivos eletrônicos através da combinação de blocos gráficos pré-programados que realizam funções específicas, sem a necessidade de escrever código. A plataforma é compatível com diversos microcontroladores populares, como ESP8266 e ESP32, e oferece uma interface gráfica amigável para facilitar a criação de programas. Para atender às necessidades de usuários já familiarizados com bancos de dados, foram desenvolvidos blocos para envio de dados da ESP32 para um banco de dados MySQL. Esses blocos foram criados para auxiliar na coleta de dados analógicos e digitais lidos pela ESP32 e no envio desses dados para um banco de dados MySQL por meio de um código PHP que faz a comunicação e o tratamento dos dados. Com a utilização desses blocos, é possível coletar dados em tempo real da ESP32 e armazená-los em um banco de dados, permitindo que esses dados sejam analisados posteriormente e utilizados para diversos fins. Para validar a eficácia dos blocos para envio de dados desenvolvidos no BIPES, foi utilizado um sensor DHT11 conectado a uma placa ESP32. O sensor foi utilizado para coletar dados de temperatura e umidade do ambiente, e os blocos foram configurados para enviar esses dados para um banco de dados MySQL. Ao longo do processo de validação, foi possível confirmar a funcionalidade dos blocos e a correta comunicação entre a ESP32 e o banco de dados, permitindo a coleta e armazenamento dos dados em tempo real. A validação dos blocos foi uma etapa importante para garantir a confiabilidade e a eficiência do sistema de envio de dados para bancos de dados MySQL desenvolvido no BIPES.

Palavras-chave: BIPES, IoT, ESP32, low-code, MySQL.

Abstract

The trend of no-code development, also known as low-code, has gained popularity for its ability to accelerate the development process and allow people without programming experience to create solutions in a simplified way. One solution that fits this trend is the Block based Integrated Platform for Embedded Systems (BIPES), a block-based embedded systems development platform designed to make it easier to create electronic projects without the need for advanced programming knowledge. With BIPES, users can create programs to control electronic devices by combining pre-programmed graphical blocks that perform specific functions, without the need to write code. The platform is compatible with several popular microcontrollers such as ESP8266 and ESP32, and offers a user-friendly graphical interface to facilitate program creation. To meet the needs of users already familiar with databases, blocks were developed to send data from ESP32 to a MySQL database. These blocks were created to assist in collecting analog and digital data read by ESP32 and sending this data to a MySQL database through PHP code that communicates and processes the data. By using these blocks, it is possible to collect real-time data from ESP32 and store it in a database, allowing this data to be analyzed later and used for various purposes. To validate the effectiveness of the data sending blocks developed in BIPES, a DHT11 sensor connected to an ESP32 board was used. The sensor was used to collect temperature and humidity data from the environment, and the blocks were configured to send this data to a MySQL database. Throughout the validation process, it was possible to confirm the functionality of the blocks and the correct communication between ESP32 and the database, allowing for real-time data collection and storage. The validation of the blocks was an important step in ensuring the reliability and efficiency of the data sending system to MySQL databases developed in BIPES.

Keywords: BIPES, IoT, ESP32, low-code, MySQL.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A utilização de sistemas embarcados tem se tornado cada vez mais frequente em diferentes áreas, como a automação industrial, a robótica e a Internet das Coisas (IoT). Esses sistemas, por sua vez, têm se tornado cada vez mais complexos e sofisticados, com uma grande variedade de componentes integrados.

Dentro desse contexto, o BIPES apresenta-se como uma solução que pode facilitar a integração entre diversos componentes, tornando a criação de sistemas embarcados complexos mais simples. O BIPES tem como objetivo tornar a programação para sistemas embarcados mais acessível a um público mais amplo, eliminando a necessidade de conhecimentos avançados em linguagens de programação. Ele possui uma interface gráfica intuitiva, permitindo que os usuários construam códigos para sistemas embarcadas arrastando e soltando blocos na tela.

A criação de blocos de bancos de dados na plataforma pode contribuir para aprimorar ainda mais a eficiência desses sistemas, principalmente no que diz respeito ao armazenamento das informações geradas por eles.

É importante ressaltar que os novos blocos desenvolvidos para o BIPES visam atender a demanda de usuários que já possuem algum nível de familiaridade com bancos de dados, principalmente o mySQL. Esses usuários podem ser programadores ou estudantes da área de tecnologia que buscam uma maneira facilitada de coletar dados de microcontroladores e enviar para banco de dados através da utilização do BIPES. Além disso, a disponibilização desses novos blocos pode atrair novos usuários que desejam explorar essa área de conhecimento, aprimorando suas habilidades e conhecimento em banco de dados.

Uma das principais motivações para o desenvolvimento de novos blocos no BIPES para enviar dados ao banco de dados mySQL é a possibilidade de ampliar as funcionalidades da plataforma, permitindo que os usuários possam realizar projetos mais complexos e com maiores possibilidades de interação com outras tecnologias. Com esses novos blocos, será possível enviar e armazenar informações coletadas por sensores, por exemplo, em um banco de dados externo, o que pode ser útil para monitorar e analisar o desempenho de sistemas ou dispositivos em tempo real.

Ao longo deste trabalho serão analisadas as tecnologias utilizadas no desenvolvimento desses blocos, incluindo o uso de ferramentas auxiliares para estabelecer a conexão entre microcontroladores e bancos de dados.

1.1 Contextualização e problemática

A popularidade da Internet das Coisas tem crescido progressivamente em nossas rotinas, provocando significativas transformações na maneira como interagimos com nossos dispositivos cotidianos. Nesse contexto, inúmeras tecnologias emergem para possibilitar a troca de informações via internet (Concept House, 2022), como por exemplo:

- `Assistente virtual doméstico:` Dispositivos como o Amazon Echo e o Google Home são assistentes virtuais que podem ser usados para controlar dispositivos domésticos, como luzes, termostatos e eletrodomésticos, além de responder perguntas e reproduzir música.
- `Aquecedores e ar condicionado:`também chamados de termostatos inteligentes, são dispositivos que controlam automaticamente a temperatura de uma casa com base em padrões de uso e preferências do usuário, permitindo economizar energia.
- `Fechaduras e portões automáticos:`Fechaduras e portões inteligentes permitem que os usuários controlem o acesso à sua casa por meio de um aplicativo em seus smartphones, além de permitir que eles monitorem quem entra e sai.
- `Dispositivos de monitoramento:` Câmeras de segurança e sensores de movimento são exemplos de dispositivos de monitoramento doméstico IoT que permitem que os usuários monitorem suas casas remotamente.

Devido à vasta gama de possibilidades de interação com dispositivos IoT e à diversidade de tipos de dados que são coletados a todo instante, os desenvolvedores que trabalham com microcontroladores enfrentam desafios únicos em relação ao armazenamento e gerenciamento desses dados. Esses dispositivos geralmente têm recursos limitados de armazenamento e processamento, o que pode dificultar a coleta e análise de grandes quantidades de dados. Além disso, esses dispositivos geralmente estão em constante comunicação com outros dispositivos e sistemas, exigindo uma alta taxa de transferência de dados em tempo real.

1.2 Organização do trabalho

Para alcançar o objetivo proposto nesse trabalho, inicialmente no capitulo 2 de fundamentação teórica é apresentada uma introdução sobre o BIPES e as tecnologias que foram utilizadas para desenvolver blocos capazes de enviar dados para um banco de dados como Micropython, Big Data, HTTP Request e PHP.

No capítulo 3 de Materiais e Métodos, são apresentados os dispositivos e softwares utilizados para o desenvolvimento dos blocos, como o ESP32, uma placa de desenvolvimento

para sistemas embarcados, e o Google Blockly, uma ferramenta de programação visual. Também serão apresentados os procedimentos para a criação dos blocos de envio de dados.

No capitulo 4 de Resultados e Discussão, são apresentados os resultados obtidos com o desenvolvimento dos blocos, incluindo a validação dos mesmos e a análise dos dados enviados para o banco de dados. Também serão discutidas as possíveis melhorias para os blocos desenvolvidos e para a plataforma BIPES como um todo.

2 Fundamentação Teórica

2.1 BIPES

Com o objetivo de ganhar mais produtividade no desenvolvimento de código, nos últimos anos, vêm se popularizando a abordagem do desenvolvimento sem código, tecnologias conhecidas pelo termo *low-code*, tornando-se tendência em grandes fornecedoras mundiais de software como Microsoft, IBM e Oracle (BOCK; FRANK, 2021).

É nesse cenário que surge o *Block-Based Integrated Platform for Embedded Systems* (BIPES), uma plataforma open source, que dá suporte para o desenvolvimento de software embarcado para diversos tipos plataformas de prototipagem eletrônica e microcontroladores como Arduino, Raspberry Pi, ESP8266, ESP32, entre outros. Estes, tendo grande importância para o mercado atual, já que oferecem soluções de baixo custo com alta capacidade de processamento (UFCG, 2020).

Além da compatibilidade entre diversos tipos de microcontroladores, o BIPES permite a conexão com esses dispositivos de forma simplificada, sem a necessidade de instalar ferramentas de desenvolvimento, sendo uma plataforma totalmente baseada em ambiente web (JUNIOR et al., 2020).

Dadas essas funcionalidades do BIPES, temos a mais importante, a criação de software embarcado por meio de blocos. Há muitas possibilidades de blocos que podem ajudar no desenvolvimento de diversas funcionalidades desde as mais básicas como conexão wifi e leitura e escrita em portas analógicas de digitais, até as mais complexas como envio de requisições via rede e controles PID. A figura 1 mostra um exemplo de solução desenvolvida no BIPES.

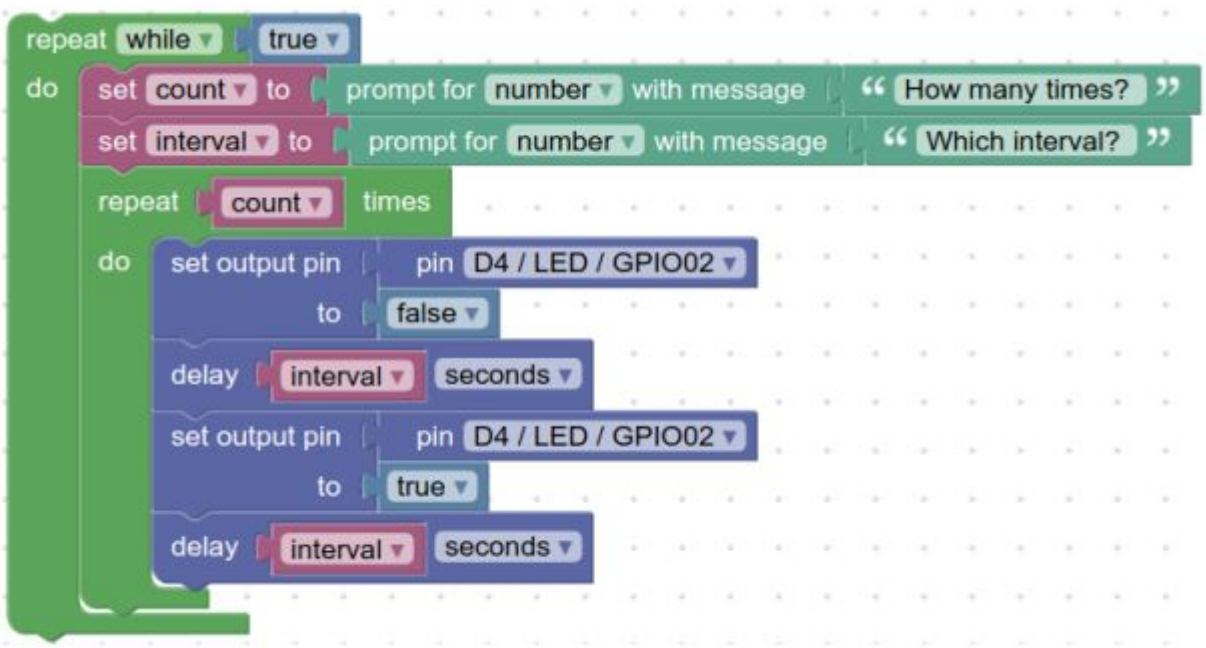
2.2 Trabalhos relacionados

Há diversos projetos com objetivos similares ao BIPES, que visam oferecer um ambiente de programação fácil de utilizar através de blocos. Algumas dessas plataformas são Open Roberta, Visualino e Blocklyduino.

- O Open Roberta [1] É uma plataforma de programação visual baseada em nuvem, desenvolvida especialmente para iniciantes em programação. O principal diferencial do Open Roberta em relação ao BIPES é o foco em programação de robôs e a integração com hardware específico, como robôs Lego Mindstorms EV3 e Calliope mini.

[1] Disponível em <lab.open-roberta.org>

Figura 1 – Exemplo de solução criada com o BIPES



Fonte: (AROCA et al., 2022)

Já o BIPES tem como objetivo principal a programação para sistemas embarcados em geral, incluindo microcontroladores como o ESP32, e oferece a possibilidade de programação de forma visual e textual. Além disso, o BIPES é uma plataforma de código aberto, o que significa que os usuários podem contribuir para o desenvolvimento da plataforma e personalizá-la de acordo com suas necessidades específicas.

- O Visualino [2] O Visualino é uma plataforma de programação visual baseada em blocos, assim como o BIPES. Ele foi desenvolvido para programação de placas Arduino. A plataforma é livre e gratuita, e oferece uma interface gráfica amigável para programação, que não requer conhecimento prévio em linguagens de programação. Assim como o BIPES o Visualino é baseado no Google blockly, porém possui a limitação de ser específico para programar em arduino, enquanto o BIPES permite a programação em diversos tipos de placas e microcontroladores como ESP8266 e ESP32.

- O Blockyduino e[3] uma plataforma para programação visual de microcontroladores Arduino, similar ao BIPES. No entanto, o BIPES apresenta algumas vantagens em relação ao Blocklyduino.

  Uma das principais vantagens do BIPES é a sua capacidade de integração com diferentes plataformas de hardware, incluindo a ESP32, que é utilizada neste trabalho.

[2] Disponível em <visualino.net>
[3] Disponível em <blocklyduino.github>

O Blocklyduino, por outro lado, é voltado especificamente para a programação de microcontroladores Arduino, limitando suas possibilidades de integração.

Além disso, o BIPES possui uma interface mais intuitiva e fácil de usar do que o Blocklyduino, o que o torna mais acessível para usuários iniciantes. O BIPES permite que os usuários criem códigos para sistemas embarcados arrastando e soltando blocos na tela, enquanto o Blocklyduino exige que os usuários montem seus programas com mais precisão, utilizando uma interface mais tradicional de programação.

O BIPES apresenta diversas vantagens em relação a outras plataformas similares. Uma das principais vantagens é a sua facilidade de uso, já que ele utiliza uma interface gráfica intuitiva para a programação de sistemas embarcados, eliminando a necessidade de conhecimentos avançados em linguagens de programação.

Além disso, o BIPES é compatível com dispositivos acessíveis e baratos, como a ESP32, o que o torna mais acessível para pessoas que desejam desenvolver projetos com menor custo. Outra vantagem é que o BIPES oferece tudo o que é necessário para desenvolver, rodar e testar aplicações em um só lugar, sem que o usuário precise baixar ou instalar nada em seu computador.

2.3 Micropython

MicroPython é uma linguagem de programação de código aberto e desenvolvimento comunitário, sendo uma adaptação do Python3 para funcionar em microcontroladores e outros ambientes restritos, e conta com um conjunto de funcionalidades da biblioteca padrão do Python.

Com relação a Hardware, o MicroPython dá suporte para muitos recursos, como interfaces de entrada e saída (por exemplo, GPIO, SPI, I2C), conversores analógico-digitais (ADCs) e comunicação protocolos como UART, WiFi, Bluetooth. Com uma implementação simplificada o MicroPython é compacto e ocupa apenas 256k de espaço de memória fash e utiliza 16k de memória RAM. (MICROPYTHON, 2022).

Essa linguagem tem como uma das principais vantagens sua simplicidade e facilidade de uso. Os desenvolvedores que já estão familiarizados com o Python podem se familiarizar rapidamente com o MicroPython, e há uma comunidade grande e ativa de desenvolvedores que contribuem para o desenvolvimento de bibliotecas e ferramentas para dar suporte a linguagem. Por conta disso o MicroPython vem sendo usado para a criação de diversas soluções no mercado de software.

Quando se trata de desenvolvimento, o MicroPython permite que os desenvolvedores escrevam código Python que pode ser executado diretamente em microcontroladores, sem a necessidade de um computador host separado ou cadeia de ferramentas complexa. Isso

torna muito mais fácil desenvolver e implantar código para sistemas embarcados e pode reduzir significativamente o tempo e o custo de desenvolvimento.

Possíveis atualizações futuras na linguagem pode trazer diversas melhorias e recursos para os usuários do BIPES, como correções de bugs, otimizações de desempenho, novas funcionalidades e suporte para novos hardwares.

Por outro lado, essa atualização pode também apresentar algumas mudanças na sintaxe e nas funcionalidades da linguagem, o que pode afetar a compatibilidade dos códigos antigos com a nova versão do MicroPython. Isso pode levar a necessidade de atualização ou reescrita de códigos antigos para que sejam compatíveis com a nova versão da linguagem, o que pode gerar um esforço adicional para os desenvolvedores do BIPES.

Com isso, temos que o Firmware MicroPython é de grande utilidade para o desenvolvimento de aplicações com microcontroladores, sendo essencial para utilização do BIPES, e será instalado na ESP32 para o desenvolvimento do módulo proposto neste trabalho.

2.4 Big Data

Em resumo, big data refere-se a um conjunto de dados que são oriundos de diversas origens, com um número muito grande de registros de alta complexidade. Dados tão complexos que os softwares tradicionais não são capazes de gerenciá-los. No entanto, esses dados podem ser usados para extrair valor, e ajuda na tomada de decisões de negócio em diversas empresas (ORACLE, 2022).

Para explicar as tecnologias de armazenamento que compõem um big data, seus conceitos são colocados em 3 pilares, sendo eles: velocidade, volume e variedade, também conhecidos como os 3 Vs do big data, para cada um temos:

- `Volume:` Refere-se a quantidade de dados, em um big data as informações coletadas são normalmente de baixa densidade como curtidas no facebook, visualizações de vídeos no instagram ou cliques em páginas web, e esses dados são sempre em grande volume, o que exige uma grande capacidade de processamento.
- `Velocidade:` Refere-se a taxa cada vez mais rápida com que os dados são recebidos, muitos sistemas operam em tempo real, armazenando os dados em memória, processando as informações coletadas e gerando resultados.
- `Variedade:` Refere-se a variedade de tipos de dados que são coletados, que comumente são dados não estruturados como textos, áudios e vídeos, o que exige um pre-processamento dessa informação para que se possa extrair algum significado deles.

A Figura 2 ilustra um resumo dessas 3 características. E é nesse cenário de big data que vamos construir nossa solução no BIPES, permitindo a coleta de informações vindas de dispositivos IoT e sendo armazenadas em uma base de dados, para que posteriormente essas informações possam ser processadas e extraídos significados delas.

Figura 2 – Resumo dos 3Vs do Big Data

Fonte: (BIGDATA, 2020)

2.5 Protocolo HTTP

HTTP é um protocolo de comunicação, que permite a trocar de informações entre um cliente e um servidor, a exemplo temos o navegador web, que atua como cliente enviando requisições para um servidor solicitando conteúdo, e o servidor responde com as informações solicitadas (SOUSA, 2018).

Para enviar ou solicitar dados para um servidor, o cliente possui alguns métodos disponíveis como GET, POST, HEAD, PUT, DELETE, entre outros. Para este trabalho vamos detalhar apenas os principais que são GET e POST:

- `GET`: Usada para solicitar um conteúdo, como por exemplo, acessar uma página web, o navegador enviar uma requisição do tipo GET e recebe as informações da página como resposta.

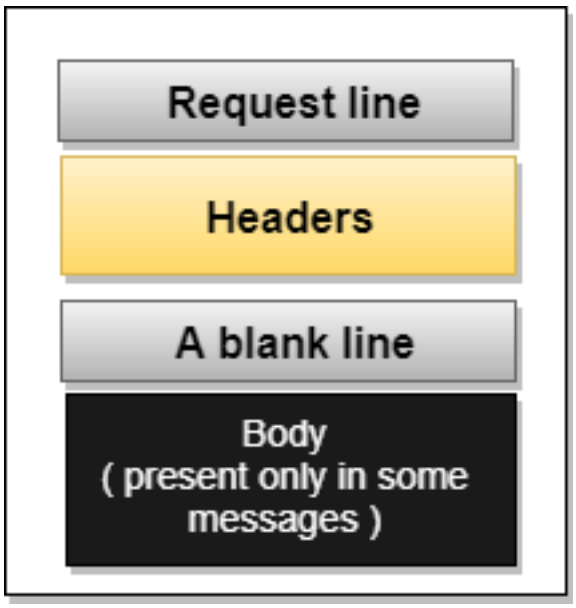
- `POST`: Usada para o cliente enviar informações ao servidor, como por exemplo, um upload de arquivo ou os dados que foram preenchidos em um formulário.

Uma requisição HTTP é composta por alguns elementos principais, que são importantes para a identificação e troca de informações entre cliente e servidor, são elas (AISANGAM, 2019):

- `Request line:` Contém 3 elementos, o método que está sendo utilizado(GET, POST, etc), a URI (*Uniform Resource Identifier*) que é o endereço eletrônico do recurso a ser utilizado e a versão do protocolo http que está sendo utilizado
- `Headers:` Contém informações adicionais a respeito da requisição e dos dados que estão sendo enviados e solicitados, como o tamanho e o tipo dos dados contidos na requisição.
- `Blanck Line:` Indica que todas informações do cabeçalho já foram inseridas, serve de separador entre o header e os dados que estão sendo enviados.
- `Body:` Usando para quando se quer transferir dados pela requisição, por exemplo, usando o método POST.

A Figura 3 mostra em alto nível os elementos que compõem uma requisição HTTP.

Figura 3 – Composição de uma requisição http



Fonte: (JAVATPOINT, 2022)

3 Materiais e Métodos

3.1 PHP

PHP: *Hypertext Preprocessor* (MATTOS; NASCIMENTO, 2009), é uma linguagem de scripts, usada para desenvolvimento web do lado do servidor, ou seja, pode ser construído dentro do servidor web. Com capacidade de geração dinâmica de conteúdo, o php pode ser usado para gerar páginas que serão visualizadas pelo lado do cliente, e também é muito eficiente quando é necessário manipular dados enviados pelo cliente. O PHP é embutido no código HTML e executado no lado do servidor, o que significa que o servidor web processa o código PHP e envia a saída que é geralmente HTML para o navegador do lado do cliente.

O PHP pode interagir com bancos de dados, executar cálculos complexos, manipular arquivos no servidor e executar uma série de outras tarefas. Ademais, o PHP é fácil de usar e oferece recursos avançados para manipulação de strings, arrays, arquivos e outras funcionalidades úteis para o processamento de dados.

Quando se trata de enviar dados da ESP32 para um banco de dados MySQL, o PHP pode ser uma escolha ideal, pois permite que você crie um script PHP simples que se conecta ao banco de dados MySQL, recebe os dados enviados pela ESP32 e os insere no banco de dados. O PHP também oferece recursos de segurança avançados, como prevenção contra ataques de injeção de SQL, que podem proteger seus dados de possíveis ameaças.

Além disso, o PHP é uma linguagem de código aberto e tem uma grande comunidade de desenvolvedores, o que significa que há muitos recursos e suporte disponíveis para ajudá-lo a trabalhar com a linguagem. Isso pode tornar o desenvolvimento mais fácil e eficiente, permitindo que se crie soluções de envio de dados para o banco de dados MySQL de forma mais rápida e com menos esforço.

3.2 ESP32

O ESP32 é um microcontrolador criado pela Espressif Systems e projetado para ser usado em aplicações de internet das coisas (IoT), sendo uma ótima opção para desenvolvimento desse tipo de solução, pois conta com uma boa capacidade de processamento e recursos para conexão wi-fi e bluetooth (ESPRESSIF, 2022).

Por ser um microcontrolador versátil o ESP32 pode ser usado para várias aplicações diferentes, podendo citar alguns exemplos:

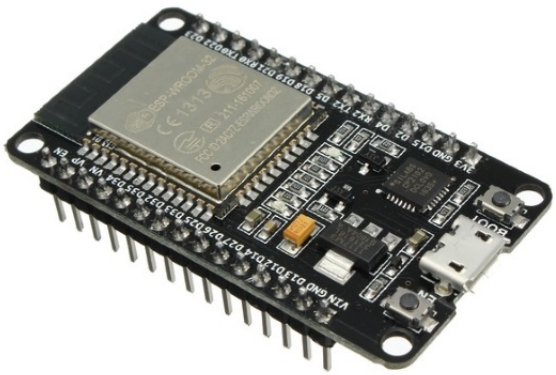
- `Automação residencial`: Com capacidade de controlar vários eletrodomésticos e

dispositivos, como luzes, ventiladores, ar condicionados e fechaduras de portas e acionamento de portões, através da conectividade Wi-Fi ou Bluetooth, o ESP32 é uma excelente opção para construir sistemas de automação residencial inteligentes.

- `Automação Industrial`: Pode ser usado na automação industrial para controlar e monitorar vários dispositivos e máquinas industriais, como bombas, motores e sensores.
- `Construção de robôs`: Pode ser usado para construir robôs e outros sistemas autônomos que necessitam de controle e comunicação usando conectividade Wi-Fi ou Bluetooth.
- `Prototipagem IoT`: : Pode ser usado por desenvolvedores para prototipagem IoT devido à sua versatilidade, alto poder de processamento e suporte para várias linguagens de programação e ambientes de desenvolvimento.

Nesse projeto foi utilizado o módulo NodeMCU ESP32 (Figura 4), que é uma placa de alta performance com microprocessador dual core de 32bits e antena wifi integrada, este dispositivo se destaca pelo baixo consumo de energia, contendo porta USB para alimentação e programação, com conversor serial integrado e possui barramento de pinos para testes em protoboard (MARTINS, 2021). A tabela 1 lista todas as especificações do módulo e a figura 5 mostra sua pinagem.

Figura 4 – NodeMCU ESP32



Fonte: (FILIPEFLOP, 2022)

Parâmetro	Valor
Alimentação:	2,2V ∼ 3,3V DC
Corrente de Consumo:	Média de 80mA
Temperatura de Operação:	-40°C ∼ +85°C
Processador:	Xtensa® Dual-Core 32-bit LX6
Processador secundário:	ULP 8MHz com consumo de 150uA.
Frequência de Operação:	80MHz 240MHz
Memoria FLASH:	4MB
Memoria RAM:	520KB
Memoria ROM:	448KB
Pinos GPIOs:	34 GPIOS de 3.3V e 12mA.
Conversores ADC :	18 ADC com 12-bit de resolução (4096 bits)
Conversores DAC :	2 ADC com 8-bit de resolução (256 bits)
WiFi:	2,4 GHz,802.11 b/g/n/e/i (802.11n até 150 Mbps)
Bluetooth:	Bluetooth Low Energy v4.2 (BLE)
Criptografia:	AES, RSA, SHA e ECC
Segurança:	WPA/WPA2/WPA2-Enterprise/WPS
Protocolos de Rede:	IPv4, IPv6, SSL, TCP/UDP/HTTP/FTP/MQTT
Temporizadores:	4 Timers de 64-bit.
Interfaces de Módulos:	UART, SPI, SDIO, I2C, LED PWM, PWM, I2S, I2C, IR

fonte: (XPROJETOS, 2020)

Tabela 1 – Especificações técnicas módulo NodeMCU ESP32

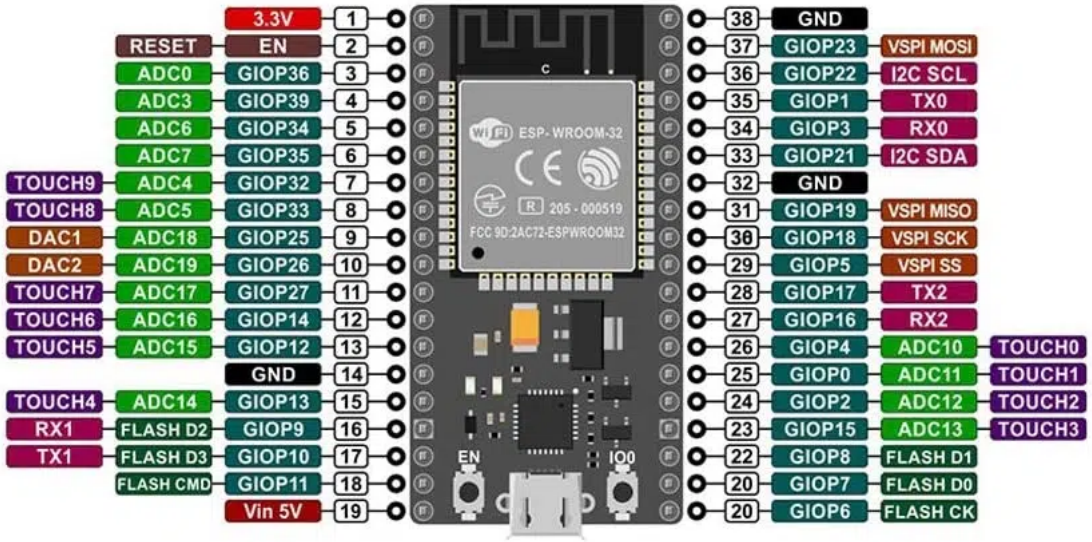
3.3 Google Blockly

O Google Blockly é uma ferramenta open-source, criada em 2012 pelo Google para o desenvolvimento de interfaces de usuário que possibilitam a criação de software por meio de blocos, sendo uma linguagem de programação visual baseada na Web projetada para tornar a programação mais acessível e intuitiva para alunos de todas as idades e origens (GOOGLE, 2023).

Com o Blockly, os usuários podem criar programas arrastando e soltando blocos que representam conceitos de programação, em vez de escrever código do zero, Os blocos podem ser empilhados verticalmente para criar programas mais complexos, com cada bloco se encaixando como uma peça de quebra-cabeça, permitindo o desenvolvimento de códigos em qualquer linguagem de programação, de maneira simples e prática.

O BIPES fornece uma integração com o Google Blockly que permite aos usuários criar programas na linguagem Python usando o editor Blockly e, em seguida, carregá-los em placas de microcontroladores compatíveis com o BIPES como ESP32, ESP8266, Arduino, Raspberry entre outras. Isso permite que os usuários programem facilmente suas placas sem precisar escrever código em uma linguagem de programação tradicional. Para criação dos blocos foi utilizado o recurso Blockly Developer Tools.

Figura 5 – Pinagem NodeMCU ESP32



Fonte: (COELHO, 2021)

3.4 Blocos desenvolvidos para envio e conexão de dados

3.4.1 Bloco para conexão com banco de dados

O processo de criação de novos blocos no BIPES é composto por algumas etapas, que são descritas na página da plataforma. Primeira etapa é criar a definição dos blocos com seus parâmetros de entrada e saída bem como sua aparência. O primeiro bloco criado, é o que vai receber todos os parâmetros de identificação e conexão do banco de dados que será usando. Utilizando o Blockly Developer Tools, foi criada a definição do bloco chamado de `db_connect` , e o código foi inserido no arquivo `block_definitions.js`:

```
1        Blockly.Blocks['db_connect'] = {
2  init: function() {
3    this.appendDummyInput()
4        .appendField("Send data to a MySQL Database")
5        .appendField(new Blockly.FieldNumber(1, 1, 9, 1), "db_idconnect"
     );
6    this.appendValueInput("db_host")
7        .setCheck("String")
8        .setAlign(Blockly.ALIGN_RIGHT)
9        .appendField("Host Addr");
10   this.appendValueInput("db_server")
11       .setCheck("String")
12       .setAlign(Blockly.ALIGN_RIGHT)
13       .appendField("Server");
14   this.appendValueInput("db_user")
15       .setCheck("String")
```

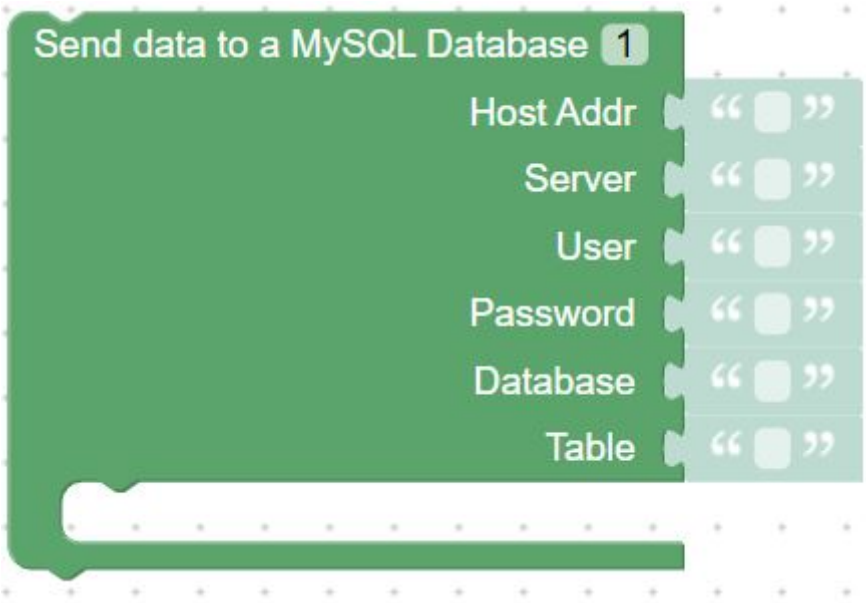
```
16          .setAlign(Blockly.ALIGN_RIGHT)
17          .appendField("User");
18      this.appendValueInput("db_pass")
19          .setCheck("String")
20          .setAlign(Blockly.ALIGN_RIGHT)
21          .appendField("Password");
22      this.appendValueInput("db_database")
23          .setCheck("String")
24          .setAlign(Blockly.ALIGN_RIGHT)
25          .appendField("Database");
26      this.appendValueInput("db_table")
27          .setCheck("String")
28          .setAlign(Blockly.ALIGN_RIGHT)
29          .appendField("Table");
30      this.appendStatementInput("db_table_data")
31          .setCheck(null)
32          .setAlign(Blockly.ALIGN_RIGHT);
33      this.setPreviousStatement(true, null);
34      this.setNextStatement(true, null);
35      this.setColour(135);
36   this.setTooltip("");
37   this.setHelpUrl("");
38    }
39 };
40
```

Seguindo este processo teremos como resultado o bloco para conexão com o banco de dados MySQL, com os parâmetros descritos abaixo, e representado no BIPES pelo bloco da Figura 6.

- `Número do bloco`: Identifica e unifica o bloco, podendo ser criados diversos blocos dentro do mesmo projeto, e enviar dados para diferentes databases.
- `Host Addr`: Identifica o endereço do Host, para onde os dados serão enviados, e posteriormente tratados.
- `Server`: Nome do servidor SQL onde os dados serão armazenados.
- `User`: Nome do usuário do banco de dados MySQL que possua permissões para inserir dados na tabela.
- `Password`: Senha do usuário do banco MySQL
- `Database`: Base de dados onde está a tabela que os dados serão inseridos.
- `Table`: Nome da tabela que os dados serão inseridos.

- `Input de dados`: Input dos dados que serão inseridos na tabela, devem ser inseridos blocos do tipo *data_value*, com o nome da coluna, tipo do dado e a informação que deve ser inserida na coluna.

Figura 6 – Bloco db_connect



Fonte: Autoria própria

3.4.2 Blocos para inserção dos dados

Para poder inserir os dados na tabela, é necessário especificar os nomes das colunas o tipo e qual dado será enviado. Com isso foi criado o bloco *data_value*, que podem ser inseridos na quantidade necessária para a aplicação do usuário. Usando o Blockly Developer Tools foi criada uma definição para este bloco e inserida no arquivo do projeto *block_definitions.js*:

```
1        Blockly.Blocks['data_value'] = {
2  init: function() {
3    this.appendDummyInput()
4        .appendField("Data type")
5        .appendField(new Blockly.FieldDropdown([["NUMBER","num"], ["
    BOOLEAN","boo"], ["TEXT","txt"], ["DATE","dat"]]), "data_type");
6    this.appendValueInput("data_column")
7        .setCheck("String")
8        .setAlign(Blockly.ALIGN_RIGHT)
9        .appendField("Column");
10   this.appendValueInput("data_value")
11       .setCheck("String")
12       .setAlign(Blockly.ALIGN_RIGHT)
13       .appendField("Value");
```

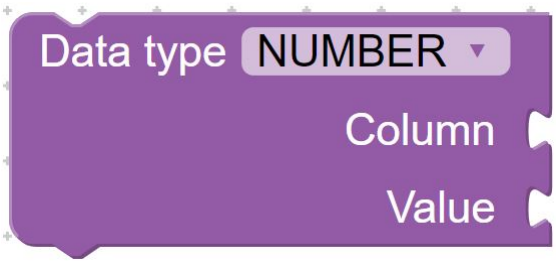
```
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(285);
 this.setTooltip("");
 this.setHelpUrl("");
  }
};

```

Seguindos este processo teremos como resultado o bloco para inserir os dados que serão enviados, com os parâmetros descritos abaixo, e representado no BIPES pelo bloco da Figura 7.

- `Data Type`: Define o tipo do dado que será enviado, sendo o campo uma *drop list* com as opções NUMBER, BOOLEAN, TEXT e DATE.
- `Column`: Nome da coluna em que o dado será inserido.
- `Value`: Dado que será inserido na coluna determinada.

Figura 7 – Bloco data_value



Fonte: Autoria própria

3.4.3 Script para geração do código MicroPython

Com os blocos criados, precisamos gerar o código em MicroPhyton que irá ser executado na ESP32, para isso, precisamos alterar o arquivo generator_stubs.js.Primeiro temos o código do bloco data_value:

```
    Blockly.Python['data_value'] = function(block) {
    var value_column = Blockly.Python.valueToCode(block, 'data_column',
    Blockly.Python.ORDER_ATOMIC);
    var value_value = Blockly.Python.valueToCode(block,
    'data_value', Blockly.Python.ORDER_ATOMIC);
    var value_type = block.getFieldValue('data_type');
    var code = value_column + ';' + value_value + ';'+ value_type;
    return code;
```

```
};

```

Esse código retorna uma string separada por ponto e vírgula, contendo os valores preenchidos no bloco. Em seguida criamos o código para o bloco db_connect, onde o código python será gerado, abaixo temos o código:

```
      Blockly.Python['db_connect'] = function(block) {
Blockly.Python.definitions_['import_urequests'] = 'import urequests';
Blockly.Python.definitions_['import_ujson'] = 'import ujson';

var number_db_idconnect = block.getFieldValue('db_idconnect');
var db_host = Blockly.Python.valueToCode(block, 'db_host', Blockly.Python.ORDER_ATOMIC);
var db_server = Blockly.Python.valueToCode(block, 'db_server', Blockly.Python.ORDER_ATOMIC);
var db_user = Blockly.Python.valueToCode(block, 'db_user', Blockly.Python.ORDER_ATOMIC);
var db_pass = Blockly.Python.valueToCode(block, 'db_pass', Blockly.Python.ORDER_ATOMIC);
var db_database = Blockly.Python.valueToCode(block, 'db_database', Blockly.Python.ORDER_ATOMIC);
var db_table = Blockly.Python.valueToCode(block, 'db_table', Blockly.Python.ORDER_ATOMIC);
var cells_blocks = block.getInputTargetBlock('db_table_data');
Blockly.Python.definitions_['post_dbdata'] = 'def post_dbdata(db_host,db_server,db_user,db_pass,db_database,db_table, db_data):\n' +
  ' request_data = ujson.dumps({"server": db_server,"user": db_user,"pass": db_pass,"database": db_database,"table": db_table,"parameters": db_data })\n'+
  ' r = urequests.post(db_host + "/" , headers = {"content-type": "application/json"}, data = request_data)\n print(r.content)\n r.close()';
  Blockly.Python.definitions_['db_host' + number_db_idconnect] = 'db_host' + number_db_idconnect + '= ' + db_host;
Blockly.Python.definitions_['db_server' + number_db_idconnect] = 'db_server' + number_db_idconnect + '= ' + db_server;
Blockly.Python.definitions_['db_user' + number_db_idconnect] = 'db_user' + number_db_idconnect + '= ' + db_user;
Blockly.Python.definitions_['db_pass' + number_db_idconnect] = 'db_pass' + number_db_idconnect + '= ' + db_pass;
Blockly.Python.definitions_['db_database' + number_db_idconnect] = 'db_database' + number_db_idconnect + '= ' + db_database;
Blockly.Python.definitions_['db_table' + number_db_idconnect] = 'db_table' + number_db_idconnect + '= ' + db_table;
Blockly.Python.definitions_['db_row_data_' + number_db_idconnect] = 'db_row_data' + number_db_idconnect +' = []';

```

```
24  if(cells_blocks)
25    var db_row_data_def = '';
26  db_row_data_def += ' global db_row_data' + number_db_idconnect + '\n';
27  db_row_data_def += ' db_row_data' + number_db_idconnect + ' = []\n';
28      do{
29        var cell_column = Blockly.Python.blockToCode(cells_blocks, '
     data_column');
30        db_row_data_def += ' db_row_data' + number_db_idconnect + ' += [
     ' +
31              '{"column": ' + cell_column.split(';')[0] + ',' +
32              ' "type": "' + cell_column.split(';')[2] + '",' +
33              ' "data": ' + cell_column.split(';')[1] + '}]' + '\n';
34
35      }while (cells_blocks = cells_blocks.getNextBlock());
36    Blockly.Python.definitions_['db_row_data_cell'+ number_db_idconnect]
     = 'def update_db_row_data'+ number_db_idconnect+'():\n' +
     db_row_data_def;
37
38  var code =  'update_db_row_data'+ number_db_idconnect +'()\n' +'
     post_dbdata(' +
39        'db_host' + number_db_idconnect+
40        ',db_server' + number_db_idconnect+
41        ',db_user' + number_db_idconnect+
42        ',db_pass' + number_db_idconnect+
43        ',db_database' + number_db_idconnect+
44        ',db_table' + number_db_idconnect+
45        ',db_row_data' + number_db_idconnect+
46        ')\n';
47        return code;
48  };
49
50
```

Essa função recebe os parâmetros do banco de dados MySQL passados no bloco e também os dados passados em todos os blocos do tipo data_value. Ponto importante de se destacar nesse código é o trecho onde as informações dos blocos data_value são tratados, para que possam ser enviadas de forma estruturada no código python, abaixo temos esse trecho:

```
1       do{
2         var cell_column = Blockly.Python.blockToCode(cells_blocks, '
     data_column');
3         db_row_data_def += ' db_row_data' + number_db_idconnect + ' += [
     ' +
4               '{"column": ' + cell_column.split(';')[0] + ',' +
5               ' "type": "' + cell_column.split(';')[2] + '",' +
```

```
            ' "data": ' + cell_column.split(';')[1] + '}]' + '\n';


    }while (cells_blocks = cells_blocks.getNextBlock());

```

3.4.4 Inclusão dos blocos na interface gráfica

O último passo de criação de novos blocos é inserir os blocos nas opções de toolbox, alterando o arquivo específico do dispositivo em que está sendo inserido o novo bloco.Como estamos implementando os novos blocos na ESP32, vamos alterar o arquivo esp32.xml. Para isso, os blocos foram inseridos na aba de Network and Internet, criando uma nova categoria, assim temos o código para o bloco db_connect:

```
<?xml version="1.0"?>
<category name="Database">
      <label text="Send data to a MySQL database"></label>
      <label text="PHP SCRIPT: https://gist.github.com"></label>
  <block type="db_connect">
    <field name="db_idconnect">1</field>
    <value name="db_host">
      <shadow type="text">
        <field name="TEXT">http://192.168.0.1/php_script.php</field>
      </shadow>
    </value>

    <value name="db_server">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="db_user">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="db_pass">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <value name="db_database">
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
     <value name="db_table">
```

```
      <shadow type="text">
        <field name="TEXT"></field>
      </shadow>
    </value>
    <statement name="db_table_data">
     <block type="data_value">
        <value name="Column">
          <shadow type="text">
            <field name="TEXT">column name</field>
          </shadow>
        </value>
        <value name="Value">
          <shadow type="text">
            <field name="TEXT">content</field>
          </shadow>
        </value>
      </block>
    </statement>

```

Em seguida foi inserido o bloco data_value, para poder ser acoplado no bloco db_connect, abaixo temos o código xml para inserção:

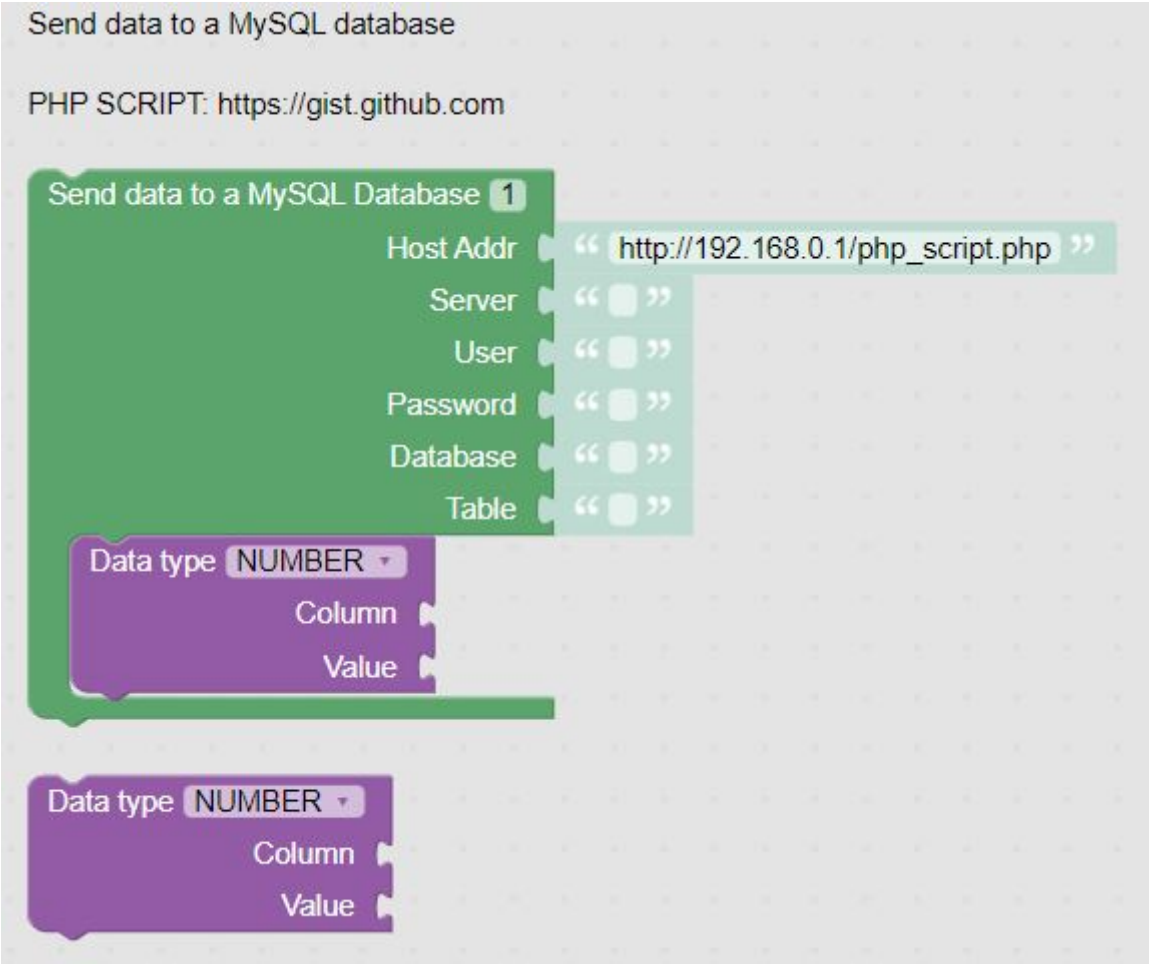
```
<?xml version="1.0"?>
</block>
   <block type="data_value">
        <value name="Column">
          <shadow type="text">
            <field name="TEXT">column name</field>
          </shadow>
        </value>
        <value name="Value">
          <shadow type="text">
            <field name="TEXT">content</field>
          </shadow>
        </value>
      </block>
```

Como resultado na interface do BIPES temos a nova categoria com os dois blocos (Figura 8), e uma máscara nos campos com pré-definição de valores, que tem o intuído de facilitar o preenchimento dos parâmetros do banco de dados.

3.4.5 Código MycroPython gerado pelos blocos

Por fim, teremos o código em Python gerado ao usar os blocos no BIPES, um exemplo com informações genéricas foi criado para facilitar a interpretação. O envio dos dados é feito através do método POST() da biblioteca urequests.py, antes disso as

Figura 8 – Categoria Database no BIPES



Fonte: Autoria própria

informações são convertidas em formato JSON para que possam ser passadas no parâmetro data. Assim temos código gerado:

```
1  import urequests
2  import ujson
3
4  def post_dbdata(db_host,db_server,db_user,db_pass,db_database,db_table,
       db_data):
5      request_data = ujson.dumps({"server": db_server,"user": db_user,"
       pass": db_pass,"database": db_database,"table": db_table,"parameters"
       : db_data })
6      r = urequests.post(db_host + "/" , headers = {"content-type": "
       application/json"}, data = request_data)
7      print(r.content)
8      r.close()
9
10 db_host1= 'http://192.168.0.9/insert_tbl.php'
11
12 db_server1= 'ServerName'
```

```
13
14 db_user1= 'UserName'
15
16 db_pass1= 'UserPassword'
17
18 db_database1= 'DatabaseName'
19
20 db_table1= 'TableName'
21
22 db_row_data1 = []
23
24 def update_db_row_data1():
25  global db_row_data1
26  db_row_data1 = []
27  db_row_data1 += [{"column": 'ColumnName1', "type": "boo", "data": '
      ColumnData1'}]
28  db_row_data1 += [{"column": 'ColumnName2', "type": "txt", "data": '
      ColumnData2'}]
29
30
31 update_db_row_data1()
32 post_dbdata(db_host1,db_server1,db_user1,db_pass1,db_database1,db_table1
    ,db_row_data1)
```

Listing 3.1 – Código Python gerado pelo bloco db_connect

4 Resultados e Discussão

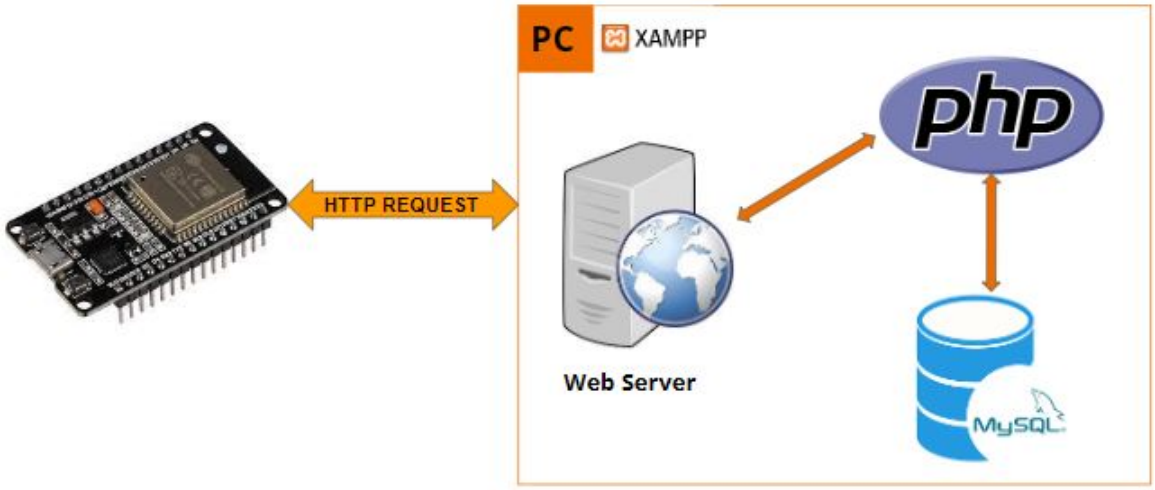
4.1 Solução usando o XAMPP

Foram desenvolvidos dois blocos que permitem enviar informações para um banco de dados MySQL através do protocolo HTTP pera rede wifi. Para demonstrar a usabilidade do bloco, foi utilizado o software XAMPP como um ambiente de desenvolvimento integrado. O XAMPP é um pacote de software livre que fornece um ambiente de servidor web completo para o desenvolvimento e teste de aplicativos web, inclui uma instalação do servidor web Apache, juntamente com um banco de dados MySQL, PHP e Perl, sendo compatível com Windows, Linux e MacOS, o que o torna uma ferramenta multiplataforma (APACHE, 2023).

O software pode ser baixado direto da página do desenvolvedor disponível no link: https://www.apachefriends.org/. Sua instalação é simples, seguindo o padrão comum de outros softwares. Também foi criada uma página no site do BIPES com os passos que serão descritos abaixo para auxiliar os usuários com o desenvolvimento, disponível no link: http://bipes.net.br/docs/block-tutorials/mysql-esp32.html.

Com o software instalado, a solução é criada e sua arquitetura completa de conexão do banco de dados e de todos as ferramentas utilizadas no processo está esquematizada na figura 9. Para receber as requisições HTTP, foi criado um `Apache Server` que é um servidor web que fornece um ambiente para hospedar aplicativos e sites da web, além de incluir recursos de segurança e ferramentas de gerenciamento e depuração.

Figura 9 – Arquitetura de transmissão de dados



Fonte: Autoria própria

Para inciar o servidor basta clicar em `START` no painel de controle do Xampp (Figura 10)

Figura 10 – Interface Xampp

Fonte: Autoria própria

4.2 Tratamento dos dados

Com o servidor iniciado, é necessário tratar os dados que serão recebidos pela requisição HTTP, para isso criamos uma aplicação PHP para realizar esse trabalho. O arquivo com extensão `.PHP` chamado de `insert_tbl.php` foi colocado na pasta `C:/xampp/htdocs/insert_tbl.php`. Ao receber uma requisição HTTP o código erá iniciar o processo de tratamento dos dados recebidos, o trecho de código abaixo mostra esse processo, os dados são decodificados usando o método `json_decode` e separados em variáveis para serem usadas em outros trechos do código:

```
1     $data = json_decode(file_get_contents('php://input'), true);
2
3 if(isset($data['parameters'])) {
4
5    $parameters = $data['parameters'];
6    $servername = $data['server'];
7    $username = $data['user'];
8    $password = $data['pass'];
9    $database_name = $data['database'];
10   $table_name = $data['table'];
```

Na variável `$parameters`, são salvos todos os dados que devem ser inseridos na tabela, no trecho a seguir os dados são separados iterativamente e construída uma string no formato que possibilite inserir os valores numa query SQL, os nomes das colunas são salvos na variável `$columns` e os valores de cada coluna são salvos na variável `$values`:

```
    for($i=0; $i<count($parameters); $i++){
      $columns .= $parameters[$i]["column"].",";

      switch ($parameters[$i]["type"]){
         case "txt":
            $values .="'".$parameters[$i]["data"]."'";
            $create_fields .=",".$parameters[$i]["column"]." VARCHAR
   (100) DEFAULT \"\"";
            break;
         case "dat":
            $dado = $parameters[1]["data"];
            array_splice($dado,3,1);
            array_splice($dado,6,1);
            $values.= "STR_TO_DATE('".implode(",",$dado)."','%Y,%m,%d,%H
   ,%i,%s')";
            $create_fields .=",".$parameters[$i]["column"]." DATETIME";
            break;
         case "boo":
            $values .= $parameters[$i]["data"];
            $create_fields .=",".$parameters[$i]["column"]." BOOL";
            break;
         default:
            $values .= $parameters[$i]["data"];
            $create_fields .=",".$parameters[$i]["column"]." DOUBLE";
      }
      $values .=",";
   }
```

A seguir temos o código completo para o processamento dos dados recebidos. Esse código foi disponibilizado na página de documentação do BIPES como parte do trabalho e pode ser utilizado como referência para enviar dados a bancos de dados MySQL em aplicações.

```
<?php

$data = json_decode(file_get_contents('php://input'), true);

if(isset($data['parameters'])) {

   $parameters = $data['parameters'];
   $servername = $data['server'];
   $username = $data['user'];
   $password = $data['pass'];
   $database_name = $data['database'];
   $table_name = $data['table'];

#cria as strings de colunas e valores para inserir na tabela
```

```
15    $columns = "(";
16    $values = "(";
17    $create_fields = "";
18
19    for($i=0; $i<count($parameters); $i++){
20       $columns .= $parameters[$i]["column"].",";
21
22       switch ($parameters[$i]["type"]){
23          case "txt":
24             $values .="'".$parameters[$i]["data"]."'";
25             $create_fields .=",".$parameters[$i]["column"]." VARCHAR
     (100) DEFAULT \"\"";
26             break;
27          case "dat":
28             $dado = $parameters[$i]["data"];
29             array_splice($dado,3,1);
30             array_splice($dado,6,1);
31             $values.= "STR_TO_DATE('".implode(",",$dado)."','%Y,%m,%d,%H
     ,%i,%s')";
32             $create_fields .=",".$parameters[$i]["column"]." DATETIME";
33             break;
34          case "boo":
35             $values .= $parameters[$i]["data"];
36             $create_fields .=",".$parameters[$i]["column"]." BOOL";
37             break;
38          default:
39             $values .= $parameters[$i]["data"];
40             $create_fields .=",".$parameters[$i]["column"]." DOUBLE";
41       }
42       $values .=",";
43    }
44
45    $columns = rtrim($columns,",");
46    $values = rtrim($values,",");
47    $columns .=")";
48    $values .=")";
49
50    // Cria uma conexao com o database
51    $connection = new mysqli($servername, $username, $password);
52    // Verifica a conexao com o database
53    if ($connection->connect_error) {
54       die("MySQL connection failed: " . $connection->connect_error);
55    }
56
57    // Check if database exists
58    $result = $connection->query("SELECT SCHEMA_NAME FROM
     INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = '$database_name'");
```

```
59
60    if (mysqli_num_rows($result) > 0) {
61        echo "Database exists <br>";
62    } else{
63        $connection->query("CREATE DATABASE $database_name");
64        echo "Database created <br>";
65
66    }
67
68    $connection = new mysqli($servername, $username, $password,
      $database_name);
69
70    $result = $connection->query("SHOW TABLES LIKE '$table_name'");
71    if (mysqli_num_rows($result) > 0) {
72        echo "Table exists <br>";
73    }else {
74    $connection->query("
75    CREATE TABLE $table_name (
76        id INT UNSIGNED NOT NULL AUTO_INCREMENT
77        $create_fields
78        ,PRIMARY KEY (id)
79    )
80    ");
81    echo "Table Created <br>";
82    }
83
84    $sql = "INSERT INTO $table_name $columns VALUES $values";
85
86    if ($connection->query($sql) === TRUE) {
87        echo "Data inserted in the table";
88    } else {
89        echo "Error: " . $sql . " => " . $connection->error;
90    }
91
92    $connection->close();
93
94 } else {
95    echo "Any data to insert";
96 }
97 ?>
```

4.3 Validação dos blocos

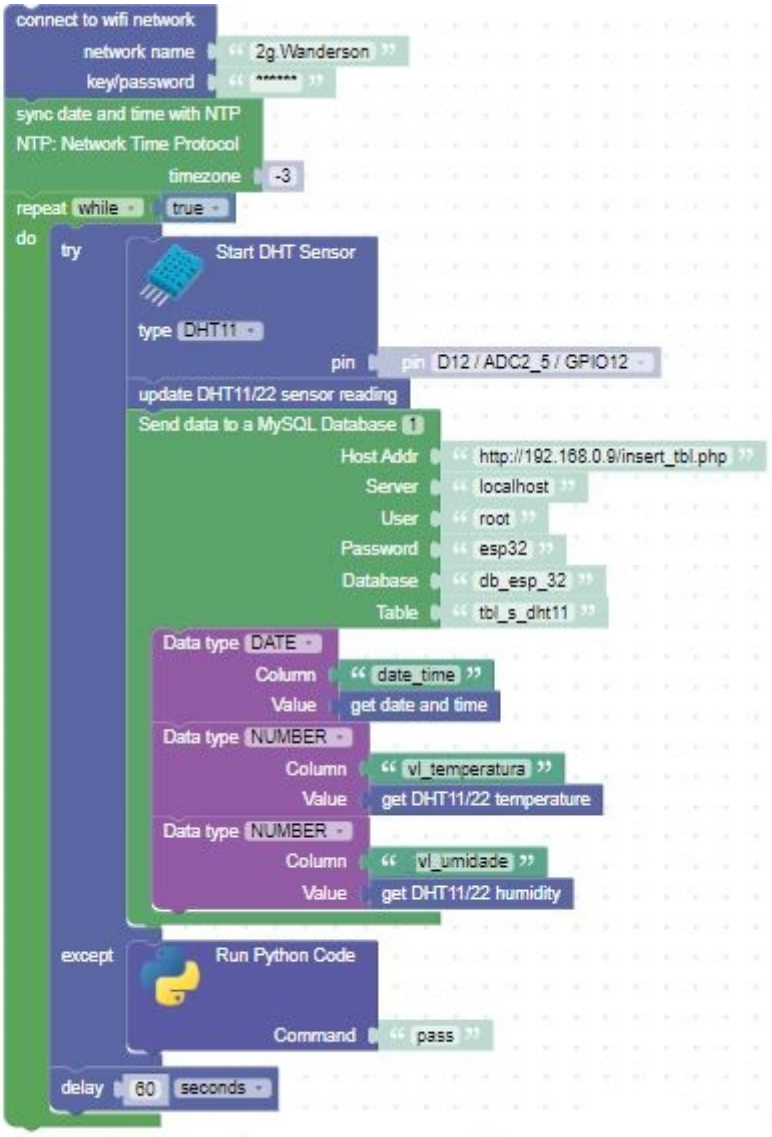
Para testar a eficácia dos blocos de envio de dados ao banco de dados, foi criado um exemplo utilizando o sensor DHT11 de temperatura e umidade. O DHT11 é um dispositivo eletrônico acessível capaz de medir com precisão a temperatura e umidade

relativa do ar em um ambiente, sendo comumente utilizado em projetos de automação residencial, sistemas de controle climático em estufas, monitoramento ambiental, entre outras aplicações.

4.3.1 Utilização dos blocos no BIPES

Para coletar os dados do sensor DHT11, foi criado uma estrutura no BIPES que permite a ESP32 conectar-se a rede wifi e operar em loop lendo os dados a cada minuto e enviando para o banco de dados MySQL, como o código de tratamento de dados em PHP cria o database e a tabela caso não existam, só há a necessidade de iniciar o banco de dados no XAMPP, a figura 11 mostra os blocos utilizados no BIPES para o exemplo.

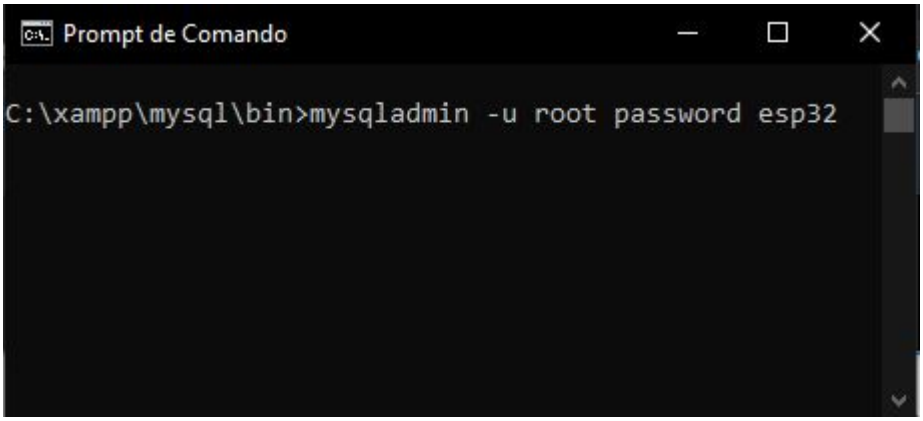
Figura 11 – Teste de validação



Fonte: Autoria própria

Com o banco de dados iniciado, é necessário configurar uma senha para o usuário `root` para acessar o banco, que por default não possui senha. Para isso, basta executar o comando `mysqladmin -u root password ROOT_PASSWORD` na pasta `C /xampp/mysql/bin` através do Prompt de Comando (Figura 12), que para nosso exemplo a senha definida foi `"esp32"`.

Figura 12 – Alteração de senha do usuário root

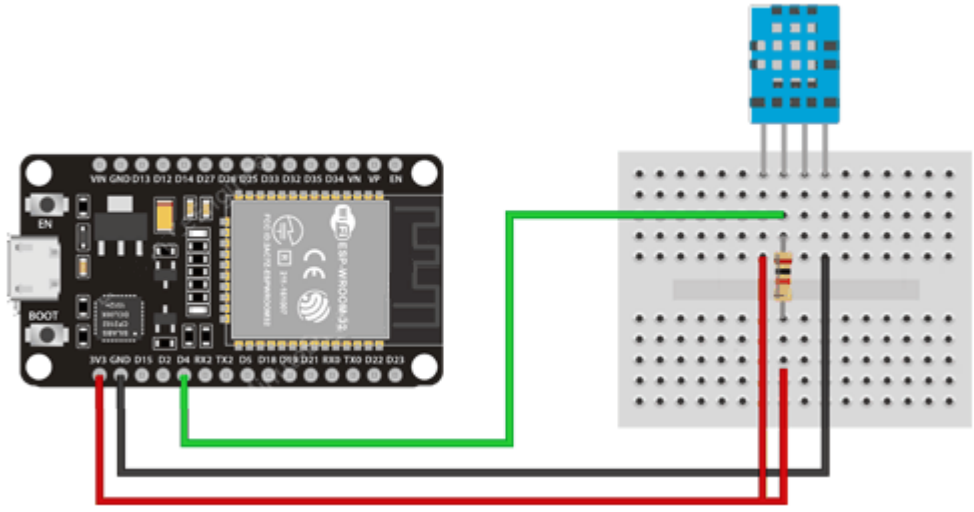


Fonte: Autoria própria

4.3.2 Circuito ESP32

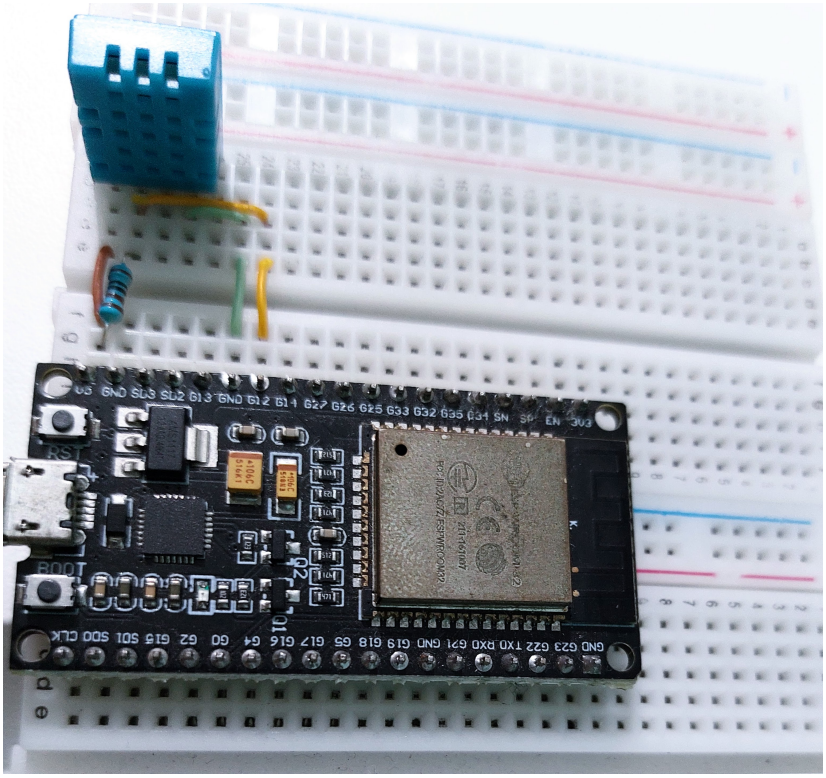
Para o funcionamento do circuito do sensor DHT11 foi utilizado uma protoboard para ligar os componentes, cabos de ligação e um resistor de 10kΩ, para melhor visualização a figura 13 mostra o esquema de ligação utilizado para ligar o sensor, e a figura 14 mostra o circuito montado fisicamente para realização dos testes.

Figura 13 – Circuito ESP32 com DHT11



Fonte: (CACPNRJ; CACPNRJ, 2020)

Figura 14 – ESP32 Conectada com DHT11

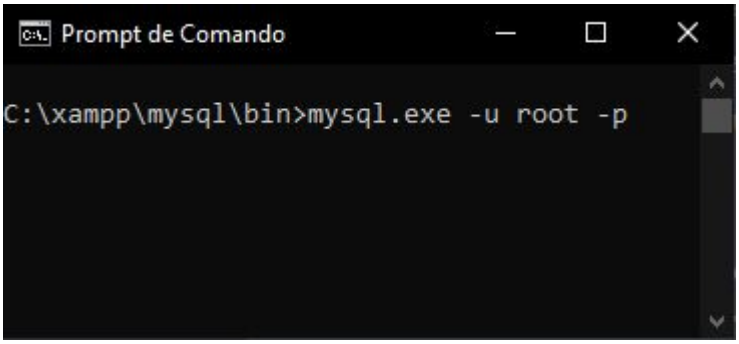


Fonte: Autoria própria

Após construir o circuito e os blocos necessários para a geração do código, o resultado obtido foi salvo no arquivo `bipes.py`. Em seguida, o arquivo foi enviado para a ESP32 através da aba `Files` da interface do BIPES, utilizando a opção `Save a copy`. Para que a ESP32 execute o código presente no arquivo `bipes.py` sempre que for iniciada, foi necessário alterar o arquivo `boot.py`, localizado na raiz do sistema de arquivos da ESP32. Para isso, foi adicionada a linha `import bipes.py` ao arquivo `boot.py` através da mesma aba `Files` do BIPES.

Deste modo, é possível conectar a ESP32 a uma fonte de energia e os dados serão automaticamente adicionados à tabela. Para visualizar os dados no banco de dados do XAMPP, basta acessar a pasta `C:/xampp/mysql/bin` no Prompt de comando e executar o comando `mysql.exe -u root -p`, conforme ilustrado na Figura 15. Em seguida, é possível acessar o terminal de comandos do banco de dados MySQL, onde é possível executar consultas e recuperar registros do banco de dados. Na Figura 16, é mostrado um exemplo de consulta, na qual é acessado o database usando o comando `use db_esp_32` e uma consulta é realizada na tabela usando o comando `select * from tbl_s_dht11`.

Figura 15 – Prompt de Comando



Fonte: Autoria própria

Figura 16 – Leitura de tabela MySQL

```
MariaDB [(none)]> use db_esp_32
Database changed
MariaDB [db_esp_32]> select * from tbl_s_dht11 limit 20;
+----+---------------------+----------------+-------------+
| id | date_time           | vl_temperatura | vl_umidade  |
+----+---------------------+----------------+-------------+
|  1 | 2023-03-21 00:40:59 |             28 |          69 |
|  2 | 2023-03-21 00:42:02 |             27 |          62 |
|  3 | 2023-03-21 00:43:05 |             27 |          63 |
|  4 | 2023-03-21 00:44:08 |             27 |          63 |
|  5 | 2023-03-21 00:45:11 |             27 |          64 |
|  6 | 2023-03-21 00:46:14 |             27 |          64 |
|  7 | 2023-03-21 00:47:17 |             27 |          64 |
|  8 | 2023-03-21 00:48:20 |             27 |          64 |
|  9 | 2023-03-21 00:49:23 |             27 |          64 |
| 10 | 2023-03-21 00:50:26 |             27 |          64 |
| 11 | 2023-03-21 00:51:28 |             27 |          64 |
| 12 | 2023-03-21 00:52:31 |             27 |          64 |
| 13 | 2023-03-21 00:53:34 |             27 |          63 |
| 14 | 2023-03-21 00:54:37 |             27 |          63 |
| 15 | 2023-03-21 00:55:40 |             27 |          63 |
| 16 | 2023-03-21 00:56:43 |             27 |          63 |
| 17 | 2023-03-21 00:57:46 |             27 |          64 |
| 18 | 2023-03-21 00:58:49 |             27 |          64 |
| 19 | 2023-03-21 00:59:52 |             27 |          64 |
| 20 | 2023-03-21 01:00:54 |             27 |          64 |
+----+---------------------+----------------+-------------+
20 rows in set (0.000 sec)
```

Fonte: Autoria própria

4.3.3 Análise dos dados

Com os dados de temperatura e umidade sendo coletados em um ambiente interno, é possível realizar diversas análises para obter insights sobre o ambiente interno da casa. alguns exemplos são:

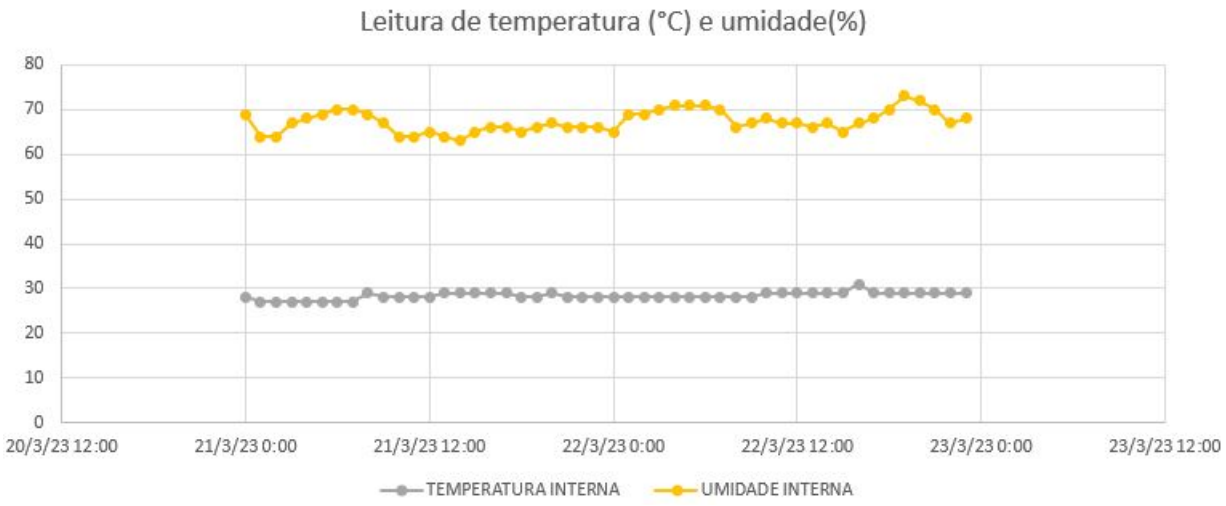
- `Identificar padrões sazonais`: Analisar os dados ao longo do tempo pode ajudar

a identificar padrões sazonais na temperatura e na umidade da casa, o que pode ser útil para planejar o uso do ar condicionado ou aquecedor.

- `Avaliar o conforto térmico`: Com os dados de temperatura e umidade, pode-se avaliar se a casa está em um nível de conforto térmico aceitável. O conforto térmico depende de vários fatores, incluindo temperatura, umidade, velocidade do ar e radiação térmica (LABEEE, 2022).
- `Identificar problemas de umidade`: os dados de umidade podem ajudar a identificar problemas de umidade na casa, como vazamentos de água ou problemas de ventilação. Isso pode ajudar a prevenir o crescimento de mofo e bolor, além de melhorar a qualidade do ar interno.

Assim, com os dados coletados de 48h seguidas, pode-se observar que o ambiente interno da residência sofre poucas variações de temperatura e umidade, tendo valores próximos em quase todos os momentos do dia, como podemos observar no gráfico gerado no excel da figura 16, temos a umidade em porcentagem com variação de 64% a 74%, com valores maiores durante a noite e menores durante o dia, e a tempera em graus Celsius variando de 27°C a 29°C.

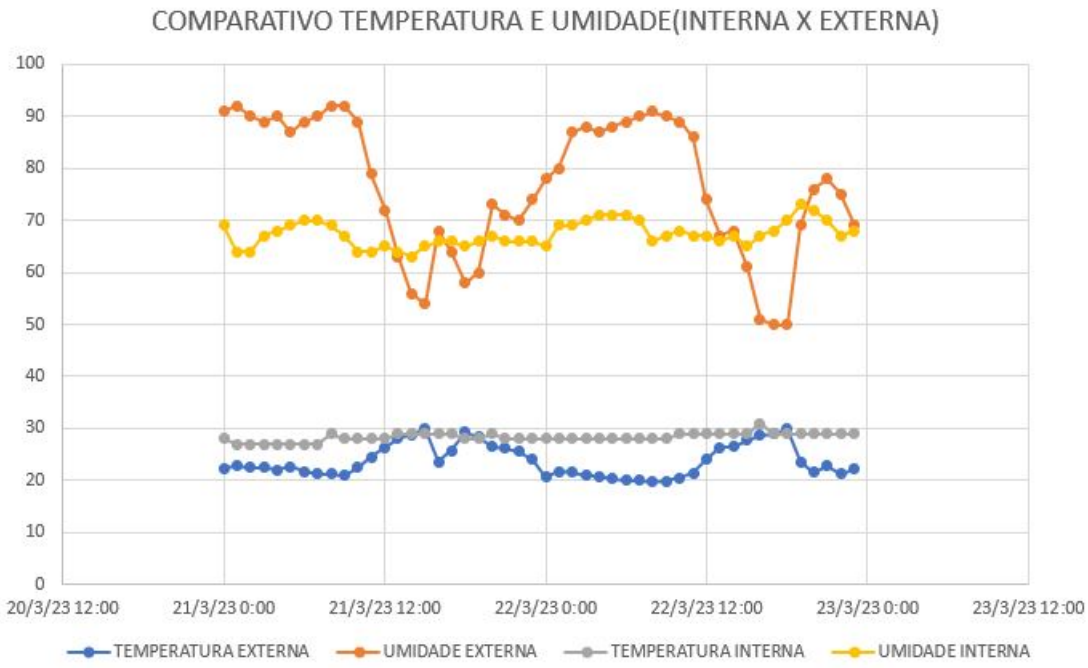
Figura 17 – Temperatura e umidade no ambiente interno



Fonte: Autoria própria, gerado no excel

Obtendo dados meteorológicos da cidade de São Carlos (INMET, 2023) nos dias de coleta de dados, podemos comparar os dados de temperatura e umidade externos com os dados coletados pelo sensor DHT11, como mostra o gráfico gerado no excel da figura 18. Assim, podemos observar que variações significativas de temperatura e umidade no ambiente externo tem pouca influência no ambiente residencial no curto período de tempo

Figura 18 – Temperatura e umidade(interna x externa)



Fonte: Autoria própria, gerado no excel

4.4 Limitações e trabalhos futuros

Com o intuito de aprimorar o desenvolvimento apresentado neste Trabalho de Graduação, é relevante enfatizar algumas possíveis melhorias e avanços que poderiam ser considerados para ampliar e aperfeiçoar a pesquisa realizada até o momento. Nessa perspectiva, este tópico tem como objetivo apresentar algumas sugestões de melhorias que podem ser implementadas no BIPES em trabalhos futuros, a fim de aprofundar o tema do banco de dados e enriquecer o trabalho acadêmico.

Abaixo seque os itens com possíveis melhorias para o trabalho:

- `Envio de dados não tabulares`: Para o desenvolvimento deste trabalho, foram criados blocos que permitem a inserção de dados em tabelas como valores numéricos, strings e datas. No entanto, os microcontroladores são capazes de fazer a leitura de outros tipos de dados, como arquivos de áudio e imagem, que são comumente classificados como dados não estruturados.

  Nesse sentido, uma sugestão viável para aprimorar a funcionalidade do BIPES seria implementar novos blocos de envio de dados que possibilitassem o envio e a manipulação desses tipos de dados não estruturados. Com essa expansão, o BIPES seria capaz de estabelecer uma conexão ainda maior com o conceito de big data, permitindo que os usuários possam explorar e analisar informações em formatos mais diversos e complexos.

Dessa forma, o sistema poderia oferecer uma capacidade ainda maior de processamento e análise de dados, atendendo às necessidades de usuários que buscam lidar com grandes volumes de informações de diferentes tipos. Além disso, ao oferecer suporte para dados não estruturados, o BIPES poderia se tornar ainda mais versátil e adaptável a diferentes contextos e demandas.

- `Manipulação dos dados:` Os blocos de código desenvolvidos têm uma função específica, que é enviar dados para o banco de dados. No entanto, é importante lembrar que um banco de dados oferece diversas possibilidades de operações que podem ser realizadas. Uma melhoria possível seria a implementação de blocos de código que permitissem a criação de outros operadores no banco de dados, como por exemplo, procedures, triggers e views.

  Procedures são rotinas que podem ser executadas no banco de dados para realizar uma ou mais operações de forma automatizada, tornando o processo mais eficiente e econômico em termos de tempo. Já as triggers são procedimentos que são executados automaticamente pelo banco de dados quando um evento específico ocorre, como por exemplo, a inserção de dados em uma tabela. Por fim, as views são uma espécie de consulta que retorna um conjunto de dados de uma ou mais tabelas de forma simplificada e mais fácil de serem lidas (JANGID, 2023).

  Ao implementar blocos no BIPES que permitam a criação desses operadores no banco de dados, é possível agregar mais funcionalidades à plataforma e tornar a manipulação dos dados mais eficiente e precisa. Além disso, a criação de procedures, triggers e views pode trazer benefícios como a melhoria da segurança dos dados e a redução de erros no processo de manipulação.

- `Outros bancos de dados:` Durante a execução do trabalho, os blocos e a aplicação PHP é capaz de enviar dados exclusivamente para bancos de dados MySQL. No entanto, há uma necessidade de expandir essa funcionalidade para permitir que esses blocos sejam utilizados para enviar dados para outros tipos de bancos de dados. Isso poderia ser realizado através da implementação de novos blocos de envio ou aprimorando os blocos já existentes para permitir conexão com outros tipos de bancos de dados. Dessa forma, haveria uma maior flexibilidade para que os dados possam ser armazenados em diferentes bancos de dados, de acordo com as necessidades do projeto do usuário. Isso pode ajudar a aumentar a eficiência e escalabilidade do projeto, pois permitiria a integração com uma variedade maior de sistemas e plataformas.

- `Implementar para outros microcontroladores:` Os blocos criados nesse trabalho foram desenvolvidos especificamente para funcionar com a ESP32, que é um microcontrolador amplamente utilizado em projetos de IoT. Contudo, uma melhoria

significativa seria a adaptação desses blocos para que possam funcionar também em outros tipos de microcontroladores, proporcionando uma maior flexibilidade para os desenvolvedores que desejam trabalhar com diferentes plataformas de hardware. Considerando que as bibliotecas utilizadas não são específicas apenas para a ESP32, torna-se possível implementar essas soluções em outros dispositivos de maneira fácil e com pouco esforço. Ao implementar esses blocos para funcionar em outros microcontroladores, seria possível expandir as possibilidades de uso desses blocos em uma variedade maior de projetos.

5 Conclusão

Este trabalho teve como propósito contribuir com a criação de novos blocos para a plataforma Block based Integrated Platform for Embedded Systems (BIPES), tendo em vista a complexidade de se realizar a comunicação entre o microcontrolador e o banco de dados para envio de dados, que exige um conhecimento mais avançado em programação e protocolos de comunicação. Nesse sentido, o desenvolvimento realizado representa um diferencial para a plataforma BIPES em relação a outras ferramentas similares, permitindo uma alternativa mais fácil e acessível para salvar dados lidos pelo microcontrolador ESP32 em um banco de dados MySQL.

A implementação desses novos blocos é uma contribuição significativa para o envio de dados para banco de dados usando a plataforma BIPES, tornando-a ainda mais completa e versátil para o desenvolvimento de projetos de sistemas embarcados. Embora ainda haja uma longa lista de melhorias que possam ser estudadas e desenvolvidas em trabalhos futuros, os testes realizados comprovaram que os novos blocos cumprem o propósito para o qual foram projetados, desenvolvendo soluções mais eficientes e acessíveis para sistemas embarcados.

Referências

AISANGAM. HTTP Request Message format well explained - AI SANGAM. *AI SANGAM*, out. 2019. Disponível em: <https://www.aisangam.com/blog/http-request-message-format-well-explained/#1>. Citado na página 25.

APACHE. *XAMPP Installers and Downloads for Apache Friends*. 2023. [Online; accessed 16. Mar. 2023]. Disponível em: <https://www.apachefriends.org/pt_br/index.html>. Citado na página 41.

AROCA, R. V. et al. *An introduction to the Internet of Things and Embedded Systems using block-based programming with BIPES and ESP8266 / ESP32*. 2022. [Online; accessed 12. Oct. 2022]. Disponível em: <https://bipes.net.br/b/download.php?file=IoT_Intro_with_BIPES_1stEd_English.pdf>. Citado na página 22.

BIGDATA. *BIGDATA: O que é Big Data | Kendoo Solutions*. 2020. [Online; accessed 19. Oct. 2022]. Disponível em: <https://solvimm.com/blog/o-que-e-big-data>. Citado na página 25.

BOCK, A. C.; FRANK, U. Low-Code Platform. *Bus. Inf. Syst. Eng.*, Springer Fachmedien Wiesbaden, v. 63, n. 6, p. 733–740, dez. 2021. ISSN 1867-0202. Citado na página 21.

CACPNRJ; CACPNRJ. Interface dht11 dht22 com esp32 e valores de exibição usando servidor web - cap sistema. *Cap Sistema*, dez. 2020. Disponível em: <https://capsistema.com.br/index.php/2020/12/03/interface-dht11-dht22-com-esp32-e-valores-de-exibicao-usando-servidor-web>. Citado na página 47.

COELHO, Í. Wifi esp32: Qual é o módulo ideal para meu projeto? - filipeflop. *FilipeFlop*, out. 2021. Disponível em: <https://www.filipeflop.com/blog/qual-modulo-wifi-esp32-e-ideal-para-meu-projeto>. Citado na página 30.

Concept House. *Casas inteligentes: a revolução da Internet das Coisas*. 2022. Acesso em: 15 de março de 2023. Disponível em: <https://concepthouse.com.br/casas-inteligentes-a-revolucao-da-internet-das-coisas>. Citado na página 18.

ESPRESSIF. *ESP32 Wi-Fi & Bluetooth MCU I Espressif Systems*. 2022. [Online; accessed 7. Nov. 2022]. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Citado na página 27.

FILIPEFLOP. *Confira o Módulo WiFi ESP32 Bluetooth - FilipeFlop*. 2022. [Online; accessed 6. Nov. 2022]. Disponível em: <https://www.filipeflop.com/produto/modulo-wifi-esp32-bluetooth>. Citado na página 28.

GOOGLE. *Google Blockly*. 2023. <https://developers.google.com/blockly>. [Online; accessed 25. Oct. 2022]. Citado na página 29.

INMET. *INMET, - Instituto Nacional de Meteorologia*. 2023. [Online; accessed 19. Mar. 2023]. Disponível em: <https://portal.inmet.gov.br>. Citado na página 50.

JANGID, G. *Stored Procedure, SQL Function, Triggers In Brief*. 2023. [Online; accessed 26. Mar. 2023]. Disponível em: <https://www.c-sharpcorner.com/blogs/about-store-proc-function-trigger-in-brif>. Citado na página 52.

JAVATPOINT. *HTTP Tutorial*. 2022. [Online; accessed 23. Oct. 2022]. Disponível em: <https://www.javatpoint.com/computer-network-http>. Citado na página 26.

JUNIOR, A. G. D. S. et al. BIPES: Block Based Integrated Platform for Embedded Systems. *IEEE Access*, IEEE, v. 8, p. 197955–197968, nov. 2020. ISSN 2169-3536. Citado na página 21.

LABEEE. *Conforto Térmico*. 2022. <https://labeee.ufsc.br/pt-br/linhas-de-pesquisa/conforto-termico>. Acesso em: 21 mar. 2023. Citado na página 50.

MARTINS, T. *Conhecendo o NodeMCU-32S ESP32*. 2021. <https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32>. Citado na página 28.

MATTOS, E. P.; NASCIMENTO, L. d. A. *PHP: Desenvolvimento web para iniciantes*. Novatec Editora, 2009. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=_xv1frKVlp8C&oi=fnd&pg=PR27&dq=PHP&ots=fzRo4Pm8LF&sig=ctqMHZuONRt_lKVnHTgfND6wLX0#v=onepage&q&f=false>. Citado na página 27.

MICROPYTHON, o. *MicroPython, Python for microcontrollers*. 2022. [Online; accessed 16. Oct. 2022]. Disponível em: <https://micropython.org>. Citado na página 23.

ORACLE. *O que é Big Data? | Oracle Brasil*. 2022. [Online; accessed 17. Oct. 2022]. Disponível em: <https://www.oracle.com/br/big-data/what-is-big-data>. Citado na página 24.

SOUSA, M. d. *O que é HTTP, request, GET, POST, response, 200, 404?* 2018. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-http-request-get-post-response-200-404>. Citado na página 25.

UFCG, I. R. O Que É Um Microcontrolador? - Capítulo Estudantil IEEE RAS UFCG. *Capítulo Estudantil IEEE RAS UFCG*, out. 2020. Disponível em: <https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador>. Citado na página 21.

XPROJETOS. *ESP32 – Especificação Técnica*. 2020. [Online; accessed 15. Nov. 2022]. Disponível em: <https://xprojetos.net/esp32-especificacao-tecnica>. Citado na página 29.