

Itens embarcados usados:

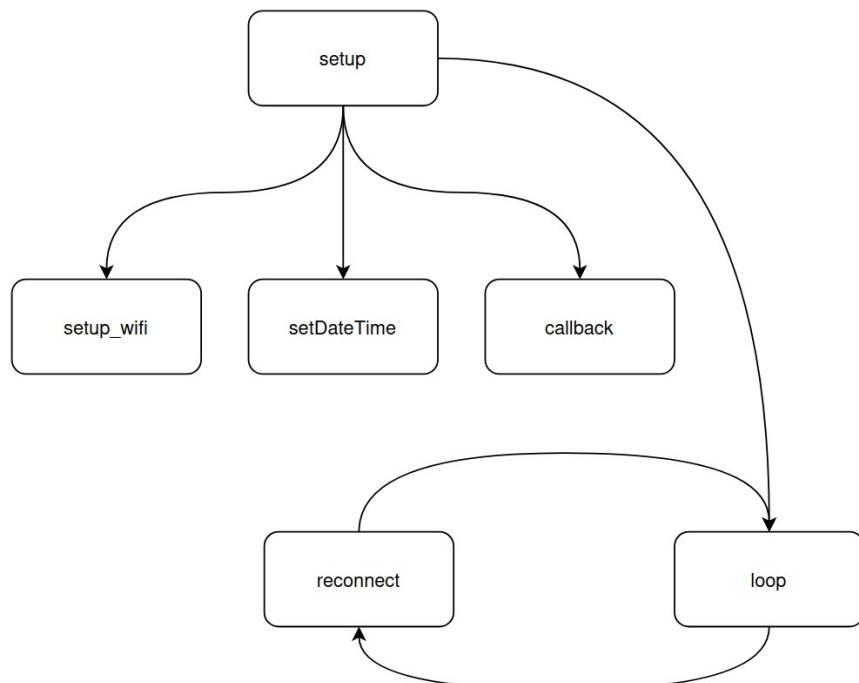
- ESP8266 NodeMCU V3
- Protoboard
- 1 Potenciometro linear de 100kΩ
- 1 Resistor de 1kΩ
- Jumpers Rígidos (ou jumpers normais)

Softwares usados:

- **Arduino IDE** (Para placa ESP8266)
- **Android Studio** para desenvolvimento do app para Android, usando a linguagem de programação Java.
- Conta no cloud **HiveMQ Cloud**, servidor MQTT do tipo serveless gratuito e seguro para conexão do ESP8266 e Adnroid para com servidor cloud.

Hardware (Embarcados)

O programa que roda no **ESP8266** é bem simples, ele segue algumas etapas para conexão segura com **Cloud**, como mostra o diagrama abaixo:



setup : Ele tem a função de inicializar o **Serial**, ler os certificados, chamar a função para configurar o wifi (**setup_wifi**), configurar horario (**setDateTime**), configurar pinos de entrada e saída, atribuir certificados para com conexão **MQTT**, configurar servidor **MQTT** e **Callback**.

setup_wifi: Ele tem a função de configurar o **WIFI**, e fica no **loop** tentando conexão até a conexão ser bem sucedida, mostrando o **IP Address**.

setDateTime: configura o fuso horário de acordo com a configuração setada, assim utiliza o **NTP** para sincronizar o horário.

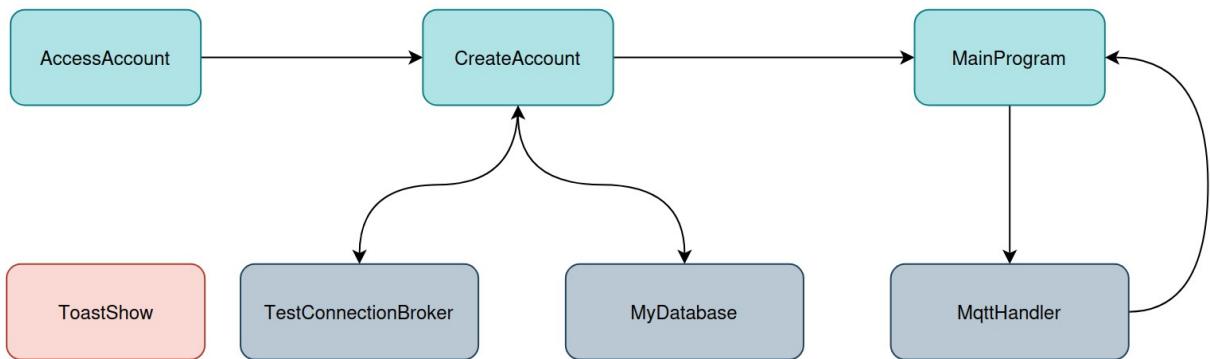
callback: Tem a função de receber as mensagens no **tópico MQTT** setado no programa, toda vez que ele recebe uma mensagem no tópico (utilizando padrão **JSON** como padronização), ele serializa e obtém os dados de **input**, se caso tenha parametro para acender o **Led**, é nessa hora que a lógica de ativação dos **Leds** ocorrem.

loop: Tem a principal função para manter o programa rodando em **loop**, checa se o cliente (no caso o **ESP8266**) está conectado com o servidor **MQTT**, se caso não esteja chama a função **reconnect**. A cada 2,5 segundos ele obtém o dado do **potenciometro** (que simula um sensor), faz a **conversao** com **analogicRead()**, serializa o **JSON** com informações do **ESP8266** (device, tempo ativo, dado do sensor, situação do Led e do LED_BUILTIN), cria um **buffer** e publica a mensagem no tópico setado.

reconnect: Tem a função de reconectar o **ESP8266** de novo ao sevidor **MQTT**, se falhar tenta de novo após **5 segundos**, se conectado publica uma mensagem no tópico avisando que está ativo e depois ler a ultima mensagem no tópico.

Software (App Android usando Java)

O programa que roda no **Android** é um pouco mais complexo do que do **ESP8266**, ele segue algumas etapas para conexão segura com **Cloud**, criação de usuário com persistência de dados usando o **MySQL Lite**, abaixo mostra o diagrama do software:



- Funções principais, que chamam telas para app (layouts)
- Funções secundárias, funcionam para regra de negócio
- Funções que podem ser chamado ao qualquer momento
(Normalmente função para exibir mensagens)

AccessAccount: Função para verificar se existe dados no banco de dados no **SQL Lite**, se caso exista exibe o layout (**access_account_layout.xml**) e o usuário pode colocar dados para login, ou criar outro usuário no botão “**Novo Usuário**”.

CreateAccount: Função para criar um novo usuário (com o layout **create_account_layout.xml**), checa os dados de **input** do usuário e valida **Username**, **TLS MQTT URL** e **Password** ao clicar no botão “**Testar Conexão**” (que chama a função **TestConnectionBroker()** como **Thread**), se caso for valido a conexão libera o botão de acesso para o **MainProgram** e salva dados de cadastro no banco de dados.

MainProgram: Tem a função principal de mostrar a tela **Home** para usuário (layout `main_program_activity.xml`) , aonde o usuário pode visualizar dados do sensor através de dados do **ESP8266** via **MQTT**, ligar ou desligar 2 leds via botões **Toggle** e chama a função principal para conexão com o **MQTT Cloud** (envio e recebimento de mensagens no tópico **MQTT**).

TestConnectionBroker: Tenta conectar com servidor **MQTT Cloud** utilizando valores do usuário no cadastro do mesmo, assim envia **Toast Message** na tela se caso conecte, salva dados no banco de dados e libera botão “**Entrar**”. Se não conectar apresenta mensagem de error via **Toast Message**.

MyDataBase: Gerencia o banco de dados **SQLite** para armazenar as credenciais do servidor **HiveMQ Cloud** do usuário. Cria a tabela **MyDataHiveMQ** para guardar **ID**, **URL**, **Username** e **Password**. Salva os dados do servidor (URL, Username, Password) no banco de dados quando **AddServerData** é chamado e envia um **Toast Message** na tela informando se os dados foram salvos com sucesso ou se ocorreu um erro. Lê os dados do banco com **GetData** e os atribui às variáveis estáticas da classe **AccessAccount**. Exclui todos os dados da tabela quando **DelData** executado.

MqttHandler: Gerencia a conexão com o **broker MQTT v5 (Paho)**. Tenta conectar usando URL, usuário e senha, e gera um **ClientID** único (**UUID**). Configura **callbacks** para lidar com eventos: ao receber uma mensagem, faz o parse do **JSON** (Sensor, Led) e envia os dados para a UI principal através de um **Handler**. Ao completar a conexão, se inscreve (ou reinscreve) automaticamente no **tópico**. Publica mensagens em uma nova **thread** para não bloquear a UI. Gerencia a desconexão e a reconexão automática.

ToastShow: Classe apenas para enviar um **Toast Message** para tela do usuário.