Utilizando WebSocket com a biblioteca AsyncWebSocket

Antes de tudo nós criamos um `objeto AsyncWebServer` na porta 80, como de costume

```
AsyncWebServer server (80);
```

A biblioteca oferece um `plugin` chamado `WebSocket` que facilita no manuseio de conexões WebSocket. Para isso criamos o objeto AsyncWebSocket, nesse caso com o nome ws, onde ws é o nosso caminho `/ws`.

```
AsyncWebSocket ws(“/ws”);
```

Lado Do Ciente Java Script

Para iniciar uma conexão Web Socket no Java script, seguimos os seguintes passos:

Descobrindo o gateway

Primeiro vamos descobrir qual o nosso gateway, e coloca-lo em uma variável junto ao caminho do websocket (/ws)

```
var gateway = ‘ws://$window.location.hostname’}/ws’;
```

`$window.location.hostname` → retorna a nossa url atual, no nosso caso o IP do esp32.

Criando o websocket

Vamos criar uma variável global chamada de websocket

```
var web socket;
```

Criamos também uma função que inicia o nosso websocket

```
function initWebSocket(){
   console.log('Abrindo a conexão websocket');
   websocket = new WebSocket(gateway);
   websocket.onopen    = onOpen;
   websocket.onclose   = onClose;
   websocket.onmessage =onMessage;
```

`console.log('Abrindo a conexão websocket');` → é opcional, apenas um feedback

`websocket = new WebSocket(gateway);` → inicia o websocket no gateway que informamos

`websocket.onopen    = onOpen;` → chama a função onOpen quando está o WebSocket está em execução

`websocket.onclose   = onClose;` → chama a função onClose quando a conexão com o web socket é fechada

`websocket.onmessage =onMessage;` → chama a função onMessage quando uma nova mensagem chega ao servidor

As funções onOpen, onClose e onMessage

Como vimos essas funções são chamadas de acordo com alguns eventos que ocorrem, agora veremos um exemplo dessas funções

```
function onOpen(event){
   console.log("Conexão iniciada");
}
```

Nesse caso a função apenas retorna um feedback

```
function onClose(event){
    console.log(“Conexão encerrada”);
    setTimeout(initWebSocket, 2000);
}
```

O console log apenas retorna um feedback

`setTimeout(initWebSocket, 2000);` → Quando estamos trabalhando com o ESP, geralmente queremos manter a conexão, então caso a conexão caia ele espera 2 segundo e conecta novamente

```
  function onMessage(event) {
    var state;
    if (event.data == "1"){
      state = "ON";
    }
    else{
      state = "OFF";
    }
    document.getElementById('state').innerHTML = state;
  }
```

Lembrando que **event**, nos trás alguns informações sobre o envento, **envent.data**, retorna o valor da menssagem

`document.getElementById('state').innerHTML = state;` → retorna ON ou OFF para o elemento com ID state

Evento load

Lembrando que o evento load ocorre quando nós o conteúdo é carregado

```
window.addEventListener('load', onLoad);
  function onLoad(event) {
    initWebSocket();
    initButton();
  }
```

A função **onLoad(event){}** apenas chama as funções initWebSocket() e initButton quando todo o conteúdo é carregado

A função initButton()

é a função que controlará o que acontece quando o botão for clicado

```
  function initButton() {
    document.getElementById('button').addEventListener('click',
toggle);
  }
```

`document.getElementById('button').addEventListener('click', toggle);` → se o botão for clicado ocorre o evento **click** e então é chamada a função **toggle()**

A função toggle()

Apenas envia uma mensagem que indica que o botão foi pressionado

```
  function toggle(){
    websocket.send('toggle');
  }
```

`websocket.send('toggle');` → o .send envia a string ‘toggle’ para o servidor

Lado Do Servidor

Como vimos anteriormente estamos utilizando a classe AsyncWebSocket da biblioteca AsyncWebServer. Dentro da classe nós temos um método chamado textAll() que notifica todos os cliente conextados com uma mensagem

```
void notifyClients(){
  ws.textAll(String(ledState));
}
```

Toda vez que a função notifyClients() for chamada, nós vamos enviar a mensagem para todos os clientes. Nesse exemplo a mensagem é o estado de um led.

A função handleWebSocketMessage()

Ela é executada sempre que recebemos novos dados do cliente via protocolo websocket

```
void handleWebSocketMessage(void *arg, uint8_t *data, size_t len)
{
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len &&
info->opcode == WS_TEXT) {
    data[len] = 0;
    if (strcmp((char*)data, "toggle") == 0) {
      ledState = !ledState;
      notifyClients();
    }
  }
}
```

`if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {data[len] = 0;}` → Verifica a integridade da mensagem, ou seja, se todos os dados chegaram corretamente

```
if (strcmp((char*)data, "toggle") == 0) {
      ledState = !ledState;
      notifyClients();
}
```
→ resumindo, se a mensagem “toggle” for recebida, nós alteramos o estado do led (no caso desse exemplo). E logo após chamamos a notifyClients() para notificar todos os clientes da mudança

Manipulador de eventos onEvent()

Ele é quem vai “ouvir” os eventos, para defini-lo fazemos da seguinte forma:

```
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient
*client, AwsEventType type,
 void *arg, uint8_t *data, size_t len) {
  switch (type) {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n",
client->id(), client->remoteIP().toString().c_str());
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n",
client->id());
      break;
    case WS_EVT_DATA:
      handleWebSocketMessage(arg, data, len);
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
  }
```

```
}
```

`switch (type) {` → `type` é o tipo de evento que ocorreu, ele pode ser dos tipos:

- `WS_EVT_CONNET` → quando o cliente se conecta
- `WS_EVT_DISCONNECT` → quando o cliente desconecta
- `WS_EVT_DATA` → Quando recebemos uma mensagem (pacote de dados) `do clienete`
- `WS_EVT_PONG` → É a resposta de solicitação de ping
- `WS_EVT_ERROR` → Quando um erro é exibido para o cliente

Os `case` apenas indicam o que fazer quando algum dos eventos acima ocorre

Iniciando o Protocolo WebSocket

```
void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}
```

É a função que será chamada no **setup()** para iniciar o servidor

Função cleanupClients()

É uma função utilizada no **loop(),** ela encerra a conexão com o cliente mais antigo caso o servidor estaja sobrecarregado, ela é útil pois nem sempre o browser fecha a conexão corretamente.